Implementation, Test, and Verification of a Specialized Real-Time Radio Protocol

Chip: AtMega128RFA1, Application: AutoStore

Ragnar Stuhaug

Master of Science in Electronics

Submission date:	July 2013
Supervisor:	Morten Olavsbråten, IET
Co-supervisor:	Ingvar Hognaland, Hatteland Computer AS

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Preface

This thesis concludes my masters degree in electronics at the Norwegian University of Technology and Science - NTNU - in Trondheim, at the same time that it allows me to connect closer to the development department at Hatteland Computer, where I have spend most of my summers, and more, during the last few years.

The background for the thesis is Hatteland Computers need for a new radio protocol for their automated storage product, and with my focus throughout my studies being logic and programming rather than differential equations and analog design, this task was more or less a perfect fit.

The core concept of the new protocol is at the courtesy of Ingvar Hognaland, the Technical Director at Hatteland Computer and the external supervisor on this thesis. His advises and feedback has been invaluable throughout the whole process.

Thanks as well to supervisor at NTNU, Morten Olavsbråten, for advises on both practical and administrative matters throughout the semester, even if he at times was somewhat hard to get hold of.

Big thanks as well to Jørgen Heggebø, at Hatteland Computer, for supplying the needed hardware and for giving advice on how to effectively debug the modules. There is no question that this saved a lot of time during the first stages of the development.

Lastly a big thanks to my fellow students for providing fun lunch breaks, and to my wife and son for allowing me to spend both evenings and weekends engulfed in c-code and coffee.

Abstract

During this thesis a specialized radio protocol has been defined, implemented, and tested. The conceptual functionality is based on a set of time synchronized access points, all operating on a different channel within the 2.4GHz ISM band. These access points will transmit beacons with a specific time offset between them, enabling listeners to continuously cycle through all channels, receiving all beacons transmitted by all access points within range. This also enables the access points to change channels at will, making the protocol very dynamic.

When a node wish to transmit it will temporarily step out of the channel cycling sequences, and access the preferred channel using CSMA/CA. If more than one channel is available, a failure to access the best one will cause it to attempt on the second best and so on. This method will cause the traffic to automatically distribute among all available channels and access points.

The nodes measures the noise on each individual channel as they cycle through them, sending regular reports to the access points. This enables the channel swaps to be made intelligently, avoiding the most noisy channels and thus automatically adapting to any interfering networks like Wi-Fi and others.

The protocol has been implemented for an AtMega128RFA1, and the protocol core has been tested and verified to work as expected. The throughput is according to theoretical calculations and the access point is able to spontaneously swap channel with minimal impact on the overall quality of service.

Some features of the protocol did not reach the needed accuracy during the implementation phase of the thesis work, still the sum of the currently observed behavior, the bugs and problems later discovered, and the set of proposed solutions to these problems, strongly indicates that the needed accuracy is well within reach. This is particularly the case with the time synchronization of the access points and the automatic detection of interference on the currently active channel.

In total the original conceptual idea has been proven to work in a real setting, giving very good results and giving strong indications that the protocol will be very well suited for the target application. The total amount of work needed to get the full protocol, with all its features, completed proved too big for a single theses, thus the current system still needs more development, debugging, and testing before it can be used in a commercial product.

Still, as a proof of concept, and as a foundation for future work, this thesis must be said to be a great success.

Samandrag

Gjennom arbeidet med denne masteroppgåva har ein spesialisert radioprotokoll blitt definert, implementert og testa. Den konseptuelle funksjonaliteten er basert på ei samling tidssynkroniserte aksesspunkt, som alle opererer på forskjellige kanalar i 2.4GHz ISM bandet. Desse aksesspunkta vil sende "beacons" ved faste intervall, desse sendingane er så tidsforseinka ei spesifikk tid mellom kvar kanal. Dette gjer det mogeleg for ein lyttar å rullere gjennom alle kanalane, og ta i mot samtlege "beacons" frå samtlege aksesspunkt innan rekkevidde. Dette gjer det også mogeleg for aksesspunkta å bytte kanal utan varsel, noko som gjer protokollen ekstremt dynamisk.

Når ein node ynskjer å sende vil den midlertidig forlate den normale kanal rulleringa og søke tilgong til den gunstigaste kanalen ved bruk av CSMA/CA. Dersom det er fleire kanalar tilgjengeleg, og den første kanalen ikkje er ledig, vil noden prøve igjen på den nest best osv. På denne måten vil trafikken automatisk fordele seg mellom alle tilgjengelege kanalar og aksesspunkt.

Nodane vil foreta energimålingar på kvar enkelt kanal undervegs i rulleringa, og sende regelmessige rapportar til aksesspunkta. Dette gjer det mogeleg å ta intelligente val for kanalbytter, og dermed unngå støyfyllte kanalar. På denne måten kan protokollen automatisk tilpasse seg konkurrerande trafikk frå Wi-Fi og andre nettverk innan rekkevidde.

Protokollen har blitt implementert for AtMega128RFA1, og protokollkjernen har blitt testa og verifisert til å verke som tiltenkt. Datagjennomstømming er på nivå med dei teoretiske utrekningane, og at aksesspunkta bytter kanal er knapt merkbart for applikasjonen som nyttar radiolinken.

Visse funksjonar rakk ikkje å komme til eit nivå der dei fungerte fullt som tiltenkt under utviklinga, likevel gir den observerte oppførselen, dei oppdaga kodefeila, og dei føreslåtte korreksjonane, eit klart inntrykk av at ei fullt ut fungerande løysing er godt innan rekkevidde. Dette gjeld spesielt tidssynkronisering av aksesspunkta og automatisk oppdaging av støy på den aktive kanalen.

Totalt sett er den fundamentale ideen bevist å fungere i eit reelt oppsett, med svært gode resultat. Dette indikerer at den nye protokollen vil vere ein god kandidat til nytt radio system for oppdragsgivar. Den totale arbeidsmengda for å få heile protokollen, med all tilleggsfunksjonalitet, heilt ferdig, viste seg alt for stor for ei enkel masteroppgåve. Dermed gjenstår det framleis ein heil del arbeid før protokollen kan nyttast i eit kommersielt produkt.

Likevel, som eit funksjonsbevis og som base for vidare arbeid må oppgåva seiast å vere ein stor suksess.

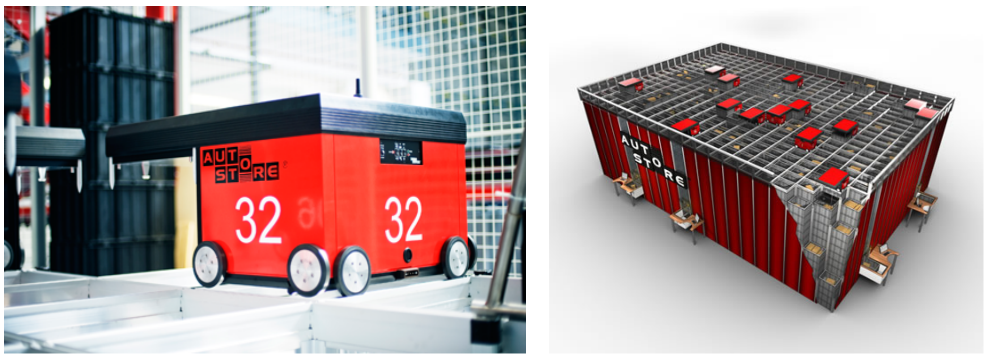
Contents

1. Introduction

The work in this thesis is directly based on the work done in [1]. In section 1.1 and section 1.2 the system presentation given in [1] is quoted directly. For convenience [1] is also included in the archive file supplied with this report.

1.1. AutoStore - Automated storage system, [1]

AutoStore is a registered trademark of the Hatteland Group and identifies a patented automated storage system [2]. Its physical structure is made up of a large aluminum grid where bins containing goods are stored on top of another in high stacks. This makes the system extremely compact compared to conventional storages using for instance shelves. On top of this grid are tracks used by a number of remote controlled robots. These robots pick up bins, move them to the desired position and take them to the ports when they are requested. The ports are where the human interaction happens. Here workers inserts goods by requesting empty bins to be filled with incoming wares, and incoming orders are executed by requesting bins containing the ordered items. By letting the system have at least a 30 minute work queue it will automatically arrange and prepare the needed bins in a manner that makes it able to keep a continuous flow of work to the operators, making the system very effective in that the user can be stationary at one location and have all the requested items delivered to that station at a very high pace.



(a) Robot (b) Example installation

Figure 1.1.: AutoStore [2]

1.2. Technical challenges and simplifications, [1]

The concept of the system makes it extremely versatile both in terms of physical layout and in terms of scale. Adding more robots to an existing system will increase its efficiency in peak cases, while the majority of the robots can be inactive in periods of little load. The grid can have any shape desired by the customer. It can even consist of multiple smaller sections connected by bridges of tracks where the robots can move freely from one section to another. This means that the radio protocol has to be able to handle a wide range of different physical topologies.

Obstacles The radio system has to be able to communicate with all robots at all times, even if it is impossible to have line of sight to all positions on the grid. There might be numerous obstacles like pillars and walls blocking the signals. Also this is likely to introduce reflections that the radio protocol has to be able to handle.

Routing Since the installation is stationary, installing multiple cabled access points is not a problem. The robots could then swap between them as needed, initiated by the system or by the radio protocol as one see fit. It's worth noting that the system will know the exact position, and the target position, of each robot at all times, and as such the need for advanced route discovery algorithms can easily be avoided. Generally there is no requirement from the system's point of view as to how the messages are routed, so long as the overall timing and throughput requirements are fulfilled.

Channels In terms of channel choices the environment will by highly static. Interfering Wi-Fi is bound to be present, but the interference will to a large degree be constant and limited to predictable frequencies. This means that in most cases one can choose, and stick to, just a few channels and avoid most, if not all, interference from other networks. However it is desired that the protocol is robust enough to handle the occurrence of interfering traffic in what was considered to be clean channels.

Emergency Stop Lastly, like all industrial systems, a failsafe emergency stop functionality is required by law. This should be incorporated with the radio protocol as a black channel concept where cutting the power of the access points should cause the robots to go into emergency stop mode. What this means is that there has to be a minimum of traffic over the channel just to indicate that the system is running.

1.3. Numerical requirements

The original requirements for the protocol as presented in [1] are shown in table 1.1. These requirements forms the foundation upon which the detailed description and behavior of the protocol is discussed. In section 2.1 multiple new aspects and considerations are presented and discussed leading up to the final protocol design.

Maximum number of clients	250
Maximum number of routers	No limit
Packet size	Median 8 bytes Max (protocol limit) bytes Min 5 bytes
Packets/second/client	Median 1 Max 7
Commands/second/client	Median 0 Max 0,33
Minimum range	30m
Latency	50% <100ms 95% <300ms 99% <500ms
Power requirement	No limit

Table 1.1.: Numerical system requirements, as stated in [1]

1.4. Research question

The task at hand is to implement, test and verify a new radio protocol to be used in an automation setting, namely the AutoStore system made by Hatteland Computer AS. The currently used protocol is based on polling and runs at 433MHz, a protocol the system can be said to have outgrown both in terms of capacity and stability. A new protocol running on the 2,4GHz ISM band is therefore to be developed. This work is a continuation of the work done in [1].

1. Implement a specialized protocol for the ATmega128RFA1 with the following properties:
   - Access points dynamically avoiding noisy channels.
   - Nodes scanning all available channels and ranking the active ones by signal strength.
   - Communication between access points and nodes through beacons.
   - Communication between nodes and access points using CSMA/CA.
2. Test and verify the functionality of the protocol on physical hardware.
3. Test "Quality of Service" in realistic situations.
   - Coexisting Wi-Fi, both on one and multiple channels.
   - Measure capacity, latency and the protocols ability to avoid noisy channels during the different tests.

2. Protocol design

2.1. Brainstorming

In [1] the proposed solution was a custom protocol based on the 802.15.4 standard [5]. By using beacons and guaranteed time slots the system would fulfill the requirements and provide a good solution. Beacons would contain messages from the access points to the nodes and the nodes would transmit their messages using CSMA/CA.

Following the release of [1] a meeting was held at Hatteland Computer's office in Nedre Vats, to discuss the findings and the future work. Participants at the meeting was the company's Technical Director, and inventor of AutoStore, Ingvar Hognaland, and the author.

During this meeting several more detailed suggestions to a protocol was made. These aspects are further discussed throughout this chapter, with the final solution presented in the final section.

2.1.1. Unacknowledged frames

It was suggested that direct acknowledgments could be skipped on messages being transmitted from the nodes to the access points. Instead the acknowledgments could be included in the beacons in the form of a list of node addresses. This approach would significantly reduce the total overhead, with the downsides being larger beacons, a maximum delay between the frame and the acknowledgement equal to the beacon period, and that the nodes would not be able to transmit more frequently than the beacon frequency. With the requirements stating that the nodes will transmit at a maximum frequency of 7Hz, any beacon frequency above this would mean that the nodes are able to transmit all the messages they want. It was assumed that it would be feasible to have beacon frequencies of anywhere from 20 to 40Hz, making the beacon period somewhere between 50 and 25 ms. With the requirements stating that the majority of the messages should be transmitted within 100ms we see that having to wait for up to 50ms for the acknowledge still puts us well within what we are able to accept.

2.1.2. Dedicated acknowledge phase

Secondly it was suggested that a period of time somewhere during the access phase following the beacons was dedicated to acknowledgments from the nodes that had received a message in the preceding beacon. This would ensure that the nodes were able to send their acknowledgments compared to if they were to use CSMA/CA. Under the 802.15.4 standard this could be done by assigning GTS'es to the nodes that received a message, wherein the node would transmit its acknowledge, and possibly also a message if it had

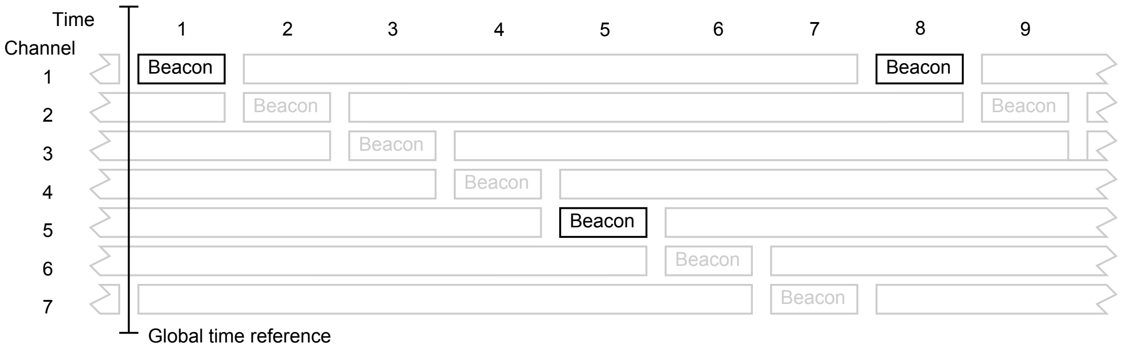


Figure 2.1.: Basic beacon sequence example. (Not to scale)

one pending. The downside of this would be that the GTS'es are bound to be at the end of the beacon period, meaning that the acknowledgement will arrive at the latest possible time. Also the standard offers limited flexibility as to the size of the GTS'es, in essence the GTS'es would be way oversized for only a single acknowledge, resulting in wasted bandwidth. A possible solution to this would be to discard the 802.15.4 standard and make a proprietary solution with GTS'es immediately following the beacon.

2.1.3. Channel agility

Lastly it was desired to make the system more dynamic and self-managing with respect to channels, with the ultimate goal to have all channel handling hidden inside the protocol itself. A solution was proposed where all the channels would be assigned a unique time offset relative to some global timing reference. Each access point would transmit its beacons following the timing offset dedicated to its assigned channel.

An idle node will, once synced to the sequence, cycle through all the channels, detecting beacons from all access points within range. A simple illustration of this is presented in figure 2.1, showing an example using seven channels and how their beacons is located sequentially in time. In this illustration channel 1 and 5 are active while the rest of the channels are unused, meaning that the nodes will listen, but not pick up any beacon, and thus not include the channel in its list of active channels. It is shown how the sequence wraps around when it reach channel 7, going back to channel 1.

The introduction of a global time reference, which would make or break this approach, was possible, and convenient, because the existing protocol used in AutoStore already has such a reference, thus only minor adjustments are needed to reuse this functionality. The existing synchronization mechanism is in the form of a 130Hz signal distributed to all the AP's from the central safety module in connection to the AP's POE supply.

With the use of this approach the need for routing is reduced significantly. Nodes will automatically detect when an AP goes in or out of range, and adjust its list of available AP's accordingly. When a node wishes to send, it will put the channel cycling on hold and lock onto the channel that is best suited for transmission, here it will transmit its message using CSMA/CA and, if the attempts succeeds, wait for the following beacon containing the acknowledge. If the CCA fails the node is free to attempt access on another channel immediately if more than one channel is available. This way the traffic load will spread among the available AP's automatically.

When the server wishes to send a message to a node it will relay the message through the AP where it last received a message from that node. This makes the routing table for the server extremely simple and self managing.

The AP's can also change channels at will without having to inform the nodes or the server in advance, the node will simply assume that one AP got out of range while another came within range, making the system extremely flexible.

One of two pitfalls with this approach is that a node might be transmitting on one channel while it is receiving a message in a beacon on a different channel. This is bound to happen when a node crosses the boundaries between two AP's and thus change what it considers as its preferred transmission channel. Also the node can simply have moved out of range of the transmitting AP. In order to overcome this problem the most important thing is to move all retransmission responsibility to the server. This way an AP that does not receive an acknowledgement for a message will simply report back to the server that the message failed. It is then up to the server to retransmit the message, on the same or on a different AP. Using this approach each lost packet from this effect can be retransmitted on the correct channel almost immediately since the channel connected to the node in the server is in most cases updated in parallel to the first transmission failing. Also it can be reasoned that, since the number of messages going from the nodes to the server is a lot larger than the number of messages going the other way (1-7 per second vs. 0-1 per three seconds as stated in table 1.1), the node is very likely to transmit a message on the new channel before the server attempts to transmit a new message in the first place, thus making the event less likely to occur.

The second pitfall is that a node might be transmitting on a channel after the AP has moved to a different one. This could be avoided by having the AP append some information to the beacon that a channel swap is imminent, preventing the node from transmitting, or one can simply accept that the packet loss from this event will occur. Assuming a beacon period of 25-50ms the node can safely retransmit the message in the next period and still be well within the 100ms latency requirement. As such this artifact could be considered an acceptable sacrifice given the benefits the overall approach offers.

2.1.4. Sequence numbers

In this scenario, as with all acknowledged communication, a seemingly failed transmission can either be caused by the original message not being transmitted correctly, or by the acknowledgement failing, making the transmission seem unsuccessful only from the senders' point of view. This is best handled by the use of sequence numbers, where the receiver checks if the message is a retransmission of a previous message or not. In this case, with the nodes moving between the AP's in the seamless manner presented in the previous section, it is clear that the AP's will not be able to know whether or not a message is a duplicate. This because any message it receives might have been attempted transmitted on a different channel before. A node can, theoretically, swap between two AP's for each transmitted packet. Because of this the sequence numbers are meaningless to include in the protocol and has to be added on a higher level by the server and the application running on the nodes. This creates the direct one to one link that is needed for sequence numbers to work.

2.1.5. Addressing

In table 1.1 it is stated that the system should be able to handle 250 independent nodes, a number implying that one byte addressing can be sufficient. However this requirement is largely related to the throughput, not the number of nodes supported by the system. As such it is highly desirable to enable addressing of more than 256 nodes. This will make sure the solution is future proof for possible extensions and changes at a later stage. The 802.15.4 standard use 16bit addresses during normal communication, and can fall back to 64bit addresses if needed, utilizing a full standard MAC address. Even if the need for 64bit addresses will most likely never be of any interest in this case, 16bits makes for a very reasonable number of nodes that should satisfy all the needs of AutoStore in the near future.

2.1.6. Extended checksum

Another consideration presented in the meeting was the need for a larger checksum than the normal 16bits used in 802.15.4. The reasons for this are previous experiences with corrupted packets being accepted as genuine commands. Though only a 1/65534 chance, the amount of messages transmitted makes even such a small chance occur multiple times per year if the radio environment is noisy and suboptimal, and the consequences of such an event can be disastrous.

It is unlikely that this problem will be as big with the new protocol as it was for a short period with this previous solution, but from that point on the choice was to be safe rather than sorry, meaning that the addition of an extra 16 bit checksum is considered a well worth overhead.

2.1.7. Network ID

In order for the channel cycling to work it is essential that the nodes are not confused by alien beacons that can bring them out of the sequence timing. This can occur if two AutoStore installations are within radio range of each other. The AP's of the two installations will not be following the same global clock and will thus be out of sync with each other. A node searching for beacons could then be at risk of locking onto the sequence of the wrong warehouse. To overcome this problem a Network ID can be introduced. This will allow for the nodes to filter out the beacons that are not of any interest to them, and it will also allow the AP's to filter out any data frames from the neighboring network.

Under normal running conditions neither event should generally happen since the AP's are to stay on different channels and, most likely, also out of sync, meaning that a synchronized node will not pick up any fake beacons and no alien node will transmit on a neighboring AP's channel. However the moment a node loose synchronization and has to search for beacons the conflict is imminent.

One challenge with this approach is how the nodes should associate with the network in the first place, as they are meant to communicate with the system purely through the radio. To overcome this one specific network id can be set as a "don't care" value where the node will sync to any network, this value will then be the default value used by the protocol. It is then up to the application to contact the server on that particular network, asking if they should associate. If this is not the case that network id could be added to a blacklist and the node forced to do a new beacon scan. When the correct network is found this network id should be stored in nonvolatile memory for future use, meaning that the association procedure only has to take place the very first time a node is added to a network.

2.1.8. Broadcasting

In addition to the normal operation of the system a special mode for broadcasting has to be supported. This mode is used to reprogram the nodes across the radio link. This involves the AP's broadcasting a large program file to all nodes, after verifying the file a boot loader takes care of loading it on the module. This mode can be completely separate from the normal operation as it will only be used when the system is in a stationary and idle condition. Also this mode is relatively straight forward as the nodes can, and should, lock onto one channel and receive the whole file from that AP. As such the implementation of this feature is not of any interest for this thesis, but it has to be taken into account when defining the protocol in order to get a unified system once the work is completed.

2.2. Proprietary or according to standard

Given the reasoning in section 2.1 the possibility of making a fully proprietary solution arises. It is obvious that a protocol as specialized as this will benefit greatly from being built from scratch without having to adjust to existing standards, also the system is not supposed to communicate with any other system or module, in fact it is desired to make such an event as difficult as possible to avoid errors. The use of 802.15.4 given the strategy described in section 2.1 is possible, but the resulting protocol will be far from optimal, and many of the features that 802.15.4 offers are not needed and will only add unnecessary overhead in this case.

In order to maximize the efficiency of the protocol it was eventually decided to attempt on a fully proprietary solution at MAC-level, but using the PHY-level described in 802.15.4. This was done based on the fact that the majority of radio SoC's support 802.15.4 and are built upon the physical description in this standard. However the functionality on the MAC level is in many cases done partly in software, meaning that the developer can easily deviate from the standard when desired.

2.3. Hardware

With respect to hardware the choice was in many ways already made as Hatteland Computer had started work with porting the existing protocol from the 433MHz band to the 2.4GHz band using the Atmel chip AtMega128RFA1 [3]. This work was started in parallel with the work done with [1] as an attempt to solve the most pressing issues with the old protocol. Since hardware had already been developed and produced in a small batch for that project, the benefits of using the same hardware for this thesis was obvious. Also the chip of choice is a very versatile and powerful chip that in any case would be a good choice.

2.3.1. Radio module

The hardware to be used consists of a Dresden Elektronik deRFmega128-22C02 [4]. This module encapsulates an AtMega128RFA1 chip and includes RF shielding, onboard EEPROM, onboard crystal oscillators and onboard coaxial antenna connector. Virtually all pins of the atmega is accessible on the outside making the chip a plug and play hardware solution while still allowing the maximum flexibility.

This module is mounted on a custom radio module specifically made for AutoStore. This chip also contains another microcontroller handling the TCP/IP communication on the AP and the communication with the rest of the hardware on the robot.

During the work with this thesis the only module of concern are the AtMega128RFA1, and the only interaction with the surrounding system is the reception of the synchronization signal. Still this is simply a pulse on one of the GPIO pins of the chip triggering an external interrupt on the AtMega128RFA1, thus no details of the surrounding system is needed to reproduce the results. In total the work and the tests done throughout the theses could have been performed on any module containing an AtMega128RFA1 and because of this no further details of the surrounding modules are included.

2.3.2. AtMega128RFA1

The AtMega128RFA1 is a complete system on chip containing a powerful microprocessor with direct access to the radio module. The radio module has an easy to use interface with control and data registers accessible directly from software. It also has several dedicated interrupts indicating all important radio events.

Transceiver

The transceiver implements the IEEE 802.15.4 physical layer modulation scheme, channel structure, physical preamble, checksum calculation and CCA. Also energy detection and link quality indication is available in hardware. On the MAC level there are dedicated modes for automated acknowledgments, CSMA/CA transmissions and retransmissions according to the standard, and automated address filtering.

Other hardware accelerators include AES 128 bit cryptography, true random number generator and PSDU rates higher than that supported in the 802.15.4 standard. Both 500 kb/s, 1 Mb/s and 2 Mb/s are supported.

In total the chip allows easy implementation of any protocol built directly upon the 802.15.4 standard, but the transceiver is controllable at a level that allows the user to deviate from the standard when needed. Among others the SFD is accessible through a dedicated register allowing the radio to differentiate its frames from 802.15.4 frames on the lowest possible "pre modulation" level.

Mac symbol counter

802.15.4 implies the use of a symbol counter to maintain the timing of the protocol, this counter should be able to wake up the transceiver before the planned reception of a frame, and it enables accurate execution of the GTS slots and CSMA/CA backoff slots as stated in the standard.

In atmega128RFA1 this counter contains three independent comparators, as well as a dedicated backoff slot counter and the possibility of automated beacon timestamping when using 802.15.4 compliant frames.

RSSI and Energy detection

When in any receiving mode the chip will continuously update a register with the current RSSI. This is a measurement of the current energy level on the channel, regardless of whether the signal is decodable or not. This value can then be read directly or the more accurate energy detection register can be used instead. The energy detection is hardware driven and calculates the average RSSI over eight symbols. The completion of this procedure is then signaled by an interrupt. In addition to the manual energy detection it is always initiated automatically upon the reception of a valid SHR, meaning that once a frame has been fully received the ED register will contain an indication of the physical signal strength that was received.

2.4. Final protocol concept

Following the requirements presented in section 1.3 and the following discussions throughout this chapter a list of features for the protocol to satisfy is presented:

- Beacons transmitted at a set frequency, with a time offset between the channels equal to the length of one such beacon.
- A global time reference for all AP's, and subsequently, nodes, to synchronize to.
- Nodes scanning all channels, receiving all beacons unless it is busy transmitting.
- Data frames transmitted using CSMA/CA in the access window of the beacon period.
- Failed CCA on one channel leads to the node attempting to access other available channels.
- Acknowledgments for data frames sent as a list of node addresses in the following beacon.
- A dedicated acknowledge phase immediately following each beacon, used by the nodes that received a message.
- No retransmissions on protocol level, all failed transmissions are reported back to the application.
- Measurement of channel quality and change of channels when needed. All concealed within the protocol.
- Broadcast mode allowing AP's to transfer larger files to all connected nodes.
- 16bit addressing of nodes, AP's only identified by their channel.
- Network ID allowing multiple installations within close proximity.
- A total of 32 bit checksum.
- Specialized frame headers to minimize overhead.

Based on this list of features a modified version of the beacon sequence from figure 2.1 is shown in figure 2.2.

The addition of a dedicated acknowledge phase will shrink the data phase slightly, but given that the acknowledgments would otherwise be transmitted within the access window, this will have no effect at all on the overall throughput.

The total access window for each active channel will be equal to the number of available channels, minus two, multiplied with the length of one subperiod. Under normal circumstances the 802.15.4 physical definition uses 16 channels in the 2.4GHz band. This means that the channel is available for transmissions 14/16 of the time, while the last 2/16 are occupied by the beacon and the acknowledge phase.

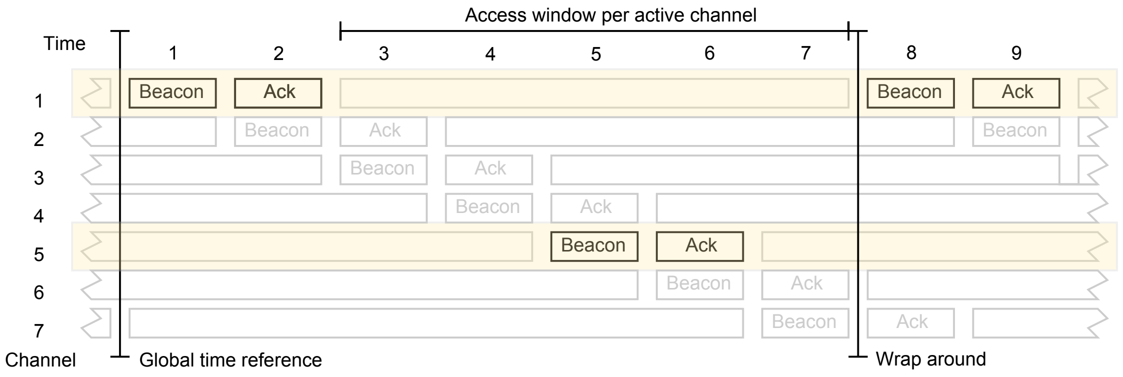


Figure 2.2.: Simple example of the time offset between channels. (Not to scale)

Since the beacons should follow a strict timing it is essential that they are always transmitting when they are supposed to, this means that there is no use in doing a CCA prior to this transmission. This lack of collision avoidance measures can cause problems for other networks in the area, Wi-Fi etc, but given the total available bandwidth in the 2.4GHz band it is a reasonable assumption that the protocol will be able to find channels that are not shared with other systems. This way the direct transmission should not pose any problems. In the event that there is a collision the worst case scenario for this protocol is that the nodes are unable to decode the beacon, leading them to believe that the AP has gotten out of range. This can trigger packet loss. Still this event, so long as it does not happen too frequently, will only cause a temporal decrease in throughput.

In order to maximize the efficiency of the acknowledge phase it is also desired to split it into dedicated slots where the nodes transmit their acknowledge frames directly. How many slots we need depends on how many messages we fit into one beacon and is further discussed in section 2.5.5.

2.5. Final protocol definition

As it is decided to deviate from the 802.15.4 standard, custom headers and frame structures has to be defined. Once this is in place the work with implementing the protocol can begin.

2.5.1. Frame header

The protocol will utilize four different frame types, summarized in table 2.1. Four different frame types can be represented using a two bit field in the header. It seems unlikely that more frame types will be needed as the types used are quite general and will most likely be reusable in most future scenarios.

The next field in the header should be the network id. It is very unlikely that more than two or three independent installations are within range of each other. However with very large installations it will be increasingly practical to split it into multiple physical grids, thus a scenario where a single installation consists of multiple independent grids is highly likely. In such a case one can imagine that 2-6, or even more, grids are within range of each other, even if they in reality are part of the same overall installation. Some

Frame	Sender	Receiver	Timing
Beacon	AP	Node	At regular intervals, time offset based on channel number enable nodes to receive beacons on all available channels.
Acknowledge	Node	AP	A node that receives a command in a beacon frame will transmit an acknowledge frame within a dedicated timeslot immediately following the beacon frame.
Data	Node	AP	Nodes transmit data frames outside the beacon/acknowledge window using CSMA/CA. The node will attempt to transmit on the perceived best channel. Acknowledge for the frame is received in the following beacon on the same channel.
Broadcast	Both	Both	Used in broadcast mode, not elaborated further in this thesis.

Table 2.1.: Frame descriptions.

Bit:	7-6	5-2	1-0
Contents:	Type	ID	Reserved

Table 2.2.: Frame header definition.

network ID's should also be reserved for special purposes, among them a "don't care" value that is accepted by all listeners. In total it is not unlikely that there might be a need for more than 8 different ID's, meaning that four bits, giving 16 different ID's should be the preferred choice.

In total that makes a 6bit header, in order to make it a full byte two extra bits are added at the end. These bits are then available for future extensions or frame specific features. For instance in connection to the channel management feature. The full header is seen in table 2.2.

2.5.2. Frame layout

The chip to be used, atmega128RFA1 [3], supports hardware driven checksum generation and verification according to the 802.15.4 standard. Also it requires the first byte in the frame buffer during transmission to be the PHR, also defined in 802.15.5. This byte consists of the frame length and a bit to indicate if the frame complies with the 802.15.4 standard or not. The frame buffer is a total of 128 byte long giving us a frame layout as seen in table 2.3. The available payload for a general frame is then a maximum of 122 bytes when checksums and other overhead are subtracted. How big the effective payload is depends on what extra overhead the different frame types introduce and are more thoroughly defined in the corresponding sections throughout this chapter.

Bytes:	(4)	(1)	1	1	1-122	2	2
Contents:	(Preamble)	(SFD)	PHR	Header	Payload	Checksum	FCS

Table 2.3.: Full frame layout, values in parenthesis are invisible to the software during transmission and reception.

Bytes:	1	1	2	1	1-119	2	2
Contents:	PHR	Header	Node ID	Data length	Payload	Checksum	FCS

Table 2.4.: Full data frame layout.

2.5.3. Data Frame

The data frame is sent by the nodes to the AP's using CSMA/CA in the access window, since only one AP will listen on the given channel it does not need any target address, but it must include the ID of the transmitting node. Also the number of data bytes is appended before the list of data begins. In total this gives the layout seen in 2.4, and as is seen here this allows for an effective payload of 119 bytes.

2.5.4. Acknowledge Frame

The acknowledge frame is sent by the node in a dedicated slot during the acknowledge phase. It is transmitted directly without any sensing of the channel in order to maintain strict timing. This frame only needs to contain the ID of the transmitting node, as this is all the AP needs to confirm that the message addressed to that particular node was received successfully. In order to keep the frame as small as possible the extra checksum is skipped, leaving only the 16bit hardware driven checksum. This can be done since the corruption of this frame is completely uncritical compared to the beacons and data frames. The AP will only look for the node ID's that it did in fact address in the previous beacon, meaning that a corrupted frame with a random ID is unlikely to be accepted by the AP even if the checksums happens to be correct. Thus the node ID itself effectively acts as a checksum. In total this gives the frame layout shown in table 2.5

2.5.5. Beacon Frame

The beacon shall contain a list of acknowledgments for the data frames received in the previous access window, and a varying number of messages addressed to different nodes. In order to optimize the functionality of the protocol the most reasonable approach will be to first add the list of acknowledgments, and then append as many messages as there are room for. This means that in high traffic situations the "AP to node"-throughput is the first to suffer, that will lead to the nodes not receiving commands as frequent as they should, which again will lead to the nodes going idle more

Bytes:	1	1	2	2
Contents:	PHR	Header	Node ID	FCS

Table 2.5.: Full ack frame layout.

frequent, thus generating less traffic "node to AP". This way the traffic flow is self regulating.

In order for the nodes to interpret the beacon correctly with a varying number of entries a beacon header is needed. This should contain the number of acknowledgments in the list and the number of following messages. This will also make it easy for nodes not expecting an acknowledgement to jump straight to the message list.

Acknowledge list

The longer the access window is, the more data frames is likely to be transmitted, and the more acknowledgments has to fit into the beacon. In table 2.9 the absolute maximum theoretical number of messages per access window is calculated assuming all transmitted messages contains 8 bytes payload. However this number is highly theoretical due to the random nature of CSMA/CA. Still it can be used as an indicator when dimensioning the fields. We see that, at a beacon frequency of 30Hz, the maximum theoretical number of data frames are 33 and at 25Hz the number increase to 40 frames.

Another factor is the maximum size of the beacon. With each acknowledge taking up two bytes, a list of 32 acknowledgments will take up 64 bytes. From table 2.9 the number of symbols available for the whole beacon phase can be found, and taking into consideration that a significant portion of these symbols will be used for overhead and synchronization between the nodes and the AP it can be assumed that the beacon frequency must be far below 30Hz for it to be able to fit more than 64 bytes or actual data.

Based on this it is clear that a maximum of 32 acknowledgments, requiring five bits to indicate the list length, should be sufficient under any realistic circumstances.

Message list

The maximum number of messages has to be evaluated in a similar manner as the number of acknowledgments. Here two matters have to be considered. Firstly the acknowledge phase has to be able to fit a number of acknowledgments equal to the maximum allowed number of messages. As stated in table 2.5 one acknowledge frame contains five bytes of data in addition to the physical header consisting of another five bytes as shown in table 2.3. In total this means that an acknowledge frame needs 20 symbols to fully transmit, in addition to this the acknowledge slots has to allow some drift, meaning that a minimum of 25 symbols should be used as a basis for any calculations. As seen in table 2.9 each subperiod will be 130 symbols long with 30Hz beacon frequency. This can allow up to five acknowledgments.

Secondly the maximum number of messages has to fit in the beacon frame. Each message will be represented by the node ID, a parameter stating the amount of data, and the data itself. Using the requirements stated in table 1.1, with average message size of 8 bytes, this means 11 bytes per message. In addition to this the requirements states that the ratio between incoming and outgoing messages from the AP's will be somewhere between 3/1 and 21/1. Meaning that each outgoing message can be assumed to be accompanied by at least three acknowledges for incoming messages. In total this means that one outgoing message from the AP can be assumed to take up an average of at least

Bit:	7	6-2	1-0
Contents:	Reserved	Ack number (0-31)	Message number (0-3)

Table 2.6.: Beacon header definition.

Bytes:	1	2	$a * 2$	$\sum_{0-m}^{i}(3 + k_i)$	2	2
Contents:	PHR	Headers	Ack list	Messages	Checksum	FCS

Table 2.7.: Full beacon frame layout.

17 bytes in the beacon, and in many cases a lot more because of a much higher number of acknowledgments.

Based on the two limitations mentioned above the message list is set to a maximum of three, making the list length possible to code with only two bits. This gives the beacon header one additional reserved bit that can be used at a later stage.

Message uniqueness

The way the protocol is designed there is nothing preventing an AP from adding multiple messages addressed to the same node to a single beacon. However that would require both the node and the AP to be able to handle the acknowledgment of all the said messages, adding unwanted complexity. The need for multiple simultaneous transmissions should never occur in the use cases the protocol is made for, and generally it is assumed that the application will concatenate the messages before adding them to the buffer, as this will save a reasonable amount of overhead. As such a simple rule is added to the protocol definitions, no beacon shall contain more than one message per node, and any broadcast message should be the only message within its beacon. This limit implies that the AP has to filter the messages to prevent any collisions, but the node will never have to wake up for more than one acknowledgement transmission.

Full beacon header

The final beacon header can be seen in table 2.6. The full beacon frame can be seen in table 2.7. Here the variable a indicates the number of acknowledgments added, m the number of messages and k_i the number of data bytes contained in message i. The sum $(a * 2) + (\sum_{0-m}^{i}(3 + k_i)) + 7$ cannot grow larger than the maximum beacon size for the current beacon frequency. This maximum size is further discussed in section 2.5.6.

2.5.6. Beacon size

The beacon frame is limited in size by the beacon frequency. In table 2.9 the symbol length of a beacon period at different frequencies can be seen. The calculations are based on 62500 802.15.4-symbols per second. One such symbol make out four bits meaning that one byte requires two symbols.

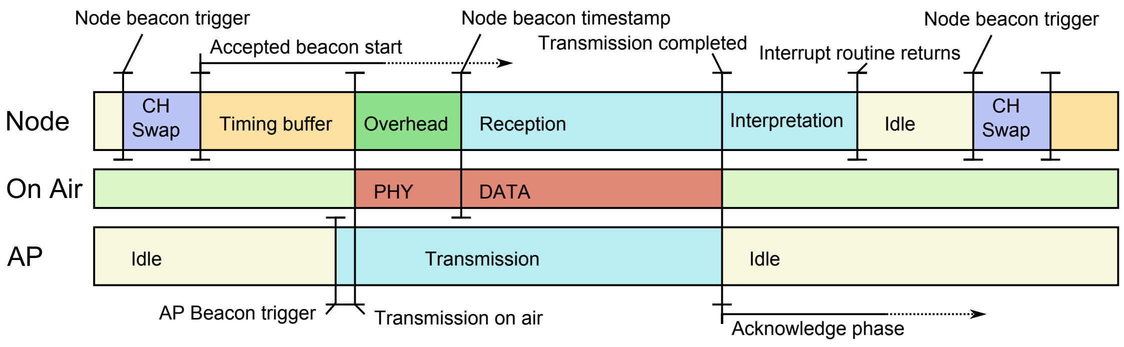


Figure 2.3.: Sketch showing the different phases of a beacon transmission.

Channel swap

In figure 2.3 a rough sketch of a beacon transmission and the connected phases are shown. In order for the node to be prepared to receive a beacon it has to adjust to the new channel before the AP starts its transmission. The time this takes has to be subtracted from the time available for beacon transmission.

According to [3] the time needed for the transceiver to settle on a new channel is a maximum of $24\mu s$. The delay on interrupts are stated to be a maximum of $9\mu s$. Since the channel swap on the node is initiated as an interrupt it can be estimated that the total time needed is $9 + 24\mu s$ + the computation time needed to initiate the swap. With a symbol length of $16\mu s$ this equates to minimum 2.1 symbols. Rounding this up to 3 symbols would then be reasonable.

AP drift

Secondly there has to be room for some timing drift between the AP's, it cannot be assumed that the AP's are perfectly synced, down to a single symbol, at all times. With a timing buffer in place two AP's can drift as far apart as the size of the buffer without causing any problems. If they drift further apart the AP that triggers too early risk triggering before the nodes has adjusted to the channel, thus the nodes will not be able to receive the beacon, and subsequently no nodes will transmit on that channel since they assume that the AP has gotten out of range or gone offline. The size of this buffer depends solely on the accuracy of the AP synchronization, meaning that it is very hard to estimate beforehand. A buffer of length 10 will allow the AP's to deviate 5 symbols in any direction from the correct timing, or $80\mu s$. If the synchronization can be made far better than this the value can be adjusted down at a later stage to allow bigger beacons.

Interpretation

Lastly the node has to be able to complete its interpretation of the beacon before the next beacon trigger, or else the beacon trigger will be delayed and the node risks getting out of sync, with disconnects as the result. The length of this period will be hard to estimate theoretically, and the best way is most likely to set up a "worst case" test procedure and find the tipping point at which it starts failing.

Phase	CH swap	Timing buffer	Overhead	Interpretation	Sum
Time	>33μs	160μs	384μs	X (160)μs	737μs
Symbols	3	10	24	X (10)	47

Table 2.8.: Total timing overhead to be subtracted from beacon period.

Freq	Subperiod size	Max beacon size (bytes)	Effective beacon size	Access window size	Max number of messages
15	260	130	106	3640	67
18	217	108	84	3038	56
20	195	97	73	2730	50
22	177	88	64	2478	45
24	162	81	57	2268	42
25	156	78	54	2184	40
26	150	75	51	2100	38
27	144	72	48	2016	37
28	139	69	45	1946	36
29	134	67	43	1876	34
30	130	65	41	1820	33
31	126	63	39	1764	32
32	122	61	37	1708	31
33	118	59	35	1652	30
34	114	57	33	1596	29
35	111	55	31	1554	28
36	108	54	30	1512	28
38	102	51	27	1428	26
40	97	48	24	1358	25

Table 2.9.: Timing and beacon calculations at different beacon frequencies.

Overhead

Once the part that does not include on air transmission is subtracted the actual overhead can be evaluated. As seen in table 2.3 a general frame has 11 bytes of overhead. Adding the beacon header makes it 12 bytes, or 24 symbols.

Note that not all this overhead is located in front of the frame, as it also includes the checksum and FCS at the end. When dimensioning timing variables and constants in the program itself this has to be taken into account.

Resulting size

In table 2.8 the timing overhead surrounding each beacon is summarized according to the phases in figure 2.3. The time needed for interpretation is set to ten initially and should be tested later to find the optimal value.

2.5.7. Checksum

The custom checksum does not need to have error correcting abilities as most corrupted frames will typically have a large degree of corruption. Therefore a simple XOR of all the bytes in the frame is used. With two bytes checksum the first is the XOR of all odd indexed bytes while the second is the XOR of all even indexed bytes. In the event that a frame contains only one byte of payload the second byte should hold its initial value, zero. Note that the PHR is not included in the checksum as this byte is not inserted into the frame buffer when a frame is received. As such the first byte to be included in the checksum is the custom frame header..

2.5.8. Synchronization

As stated in section 2.1.3 a global timing reference is introduced to maintain the strict timing requirement between the access points. The frequency of this signal must not exceed the beacon frequency. As such the existing 130Hz is unusable. For simplicity it is defined that the synchronization signal should be 1Hz. It is then up to the developers of the application to generate and supply the signal according to this requirement.

2.5.9. Functional description

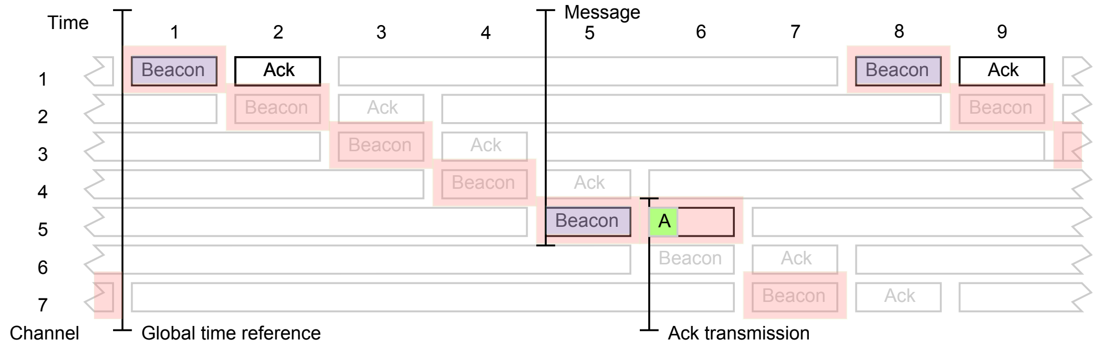
In figure 2.4 four examples of the communication sequence are shown.

In figure 2.4(a) a simple reception is shown. The node receives a message within the beacon on channel five and stays locked to that channel throughout the following acknowledge phase to transmit its acknowledge.

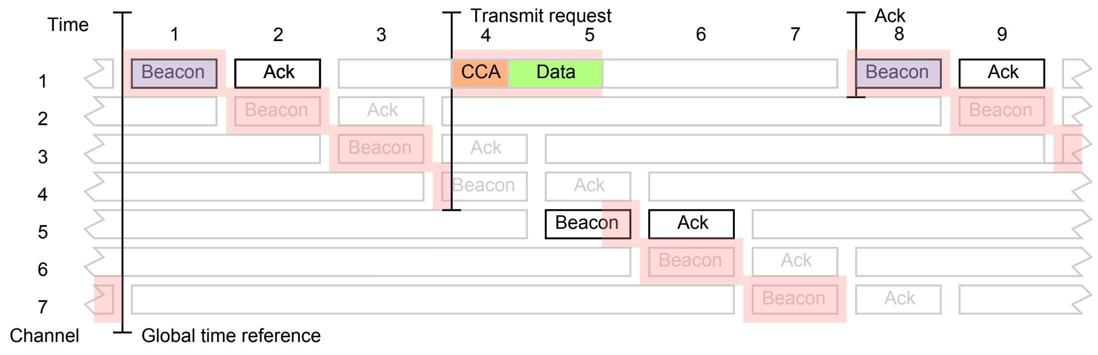
In figure 2.4(b) the node attempts to access channel one for transmission and immediately gains access. As seen the node is not able to receive the beacon on channel five, as it is transmitting on channel one at that time. This will only pose a problem if the received beacon on channel five contains a message for the node, or a broadcast. However, as discussed in section 2.1.3 the chance of the node losing a directly addressed message this way is sufficiently small for it to be an acceptable penalty.

In figure 2.4(c) the node fails its CCA on channel one, and at that point in time channel five is blocked by the beacon and acknowledge phase. Following this the node performs a random timeout and retries its transmission at a later stage.

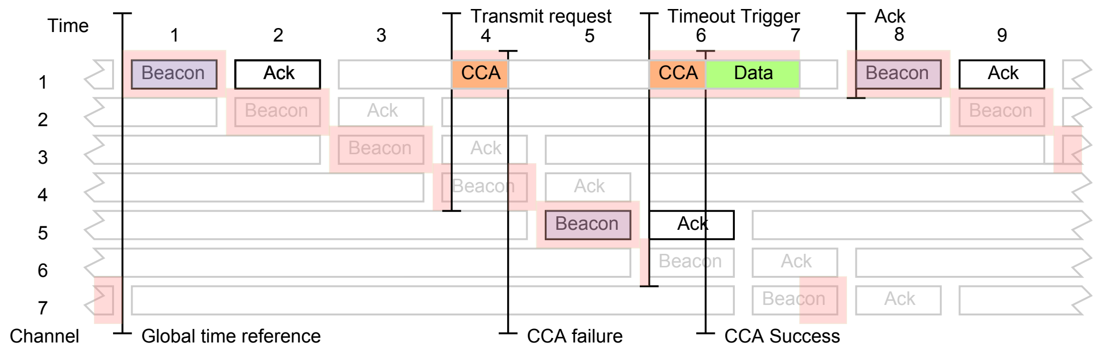
In figure 2.4(d) the node has changed the ranking of the channels and attempts to transmit on channel 5 first, this fails, but since channel one is currently available it immediately retries there, with success. Note that just like in figure 2.4(b) this cause the node to miss the beacon on channel 5.



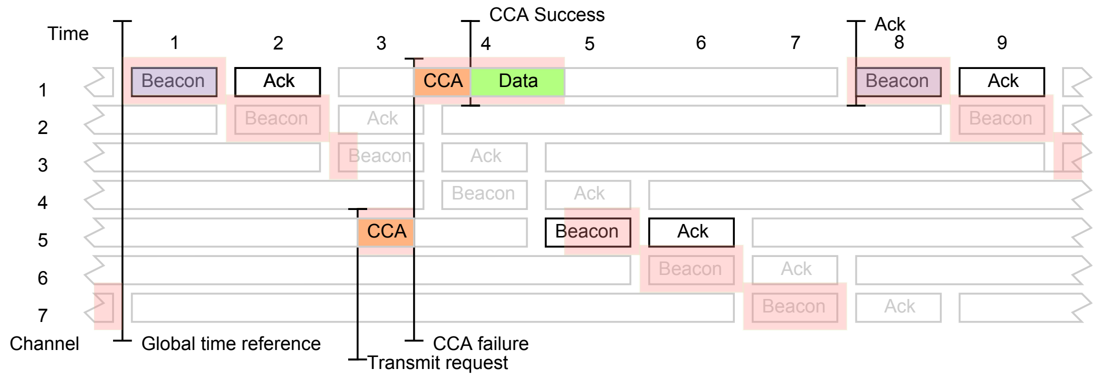
(a) Receiving message.



(b) Transmitting with immediate access to the channel.



(c) Transmitting with one failed CCA, retry on same channel since the other channel is blocked by the beacon and ack phase.



(d) Transmitting with one failed CCA, immediate retry on other channel succeeds.

Figure 2.4.: Illustrations of transmission and reception for a node. Red squares show what channel the node is linked too. Blue show when the node is actively receiving. Orange is CCA measurement of the channel. Green is node transmissions. (Not to scale)

3. Implementation

In this chapter the source code produced for the protocol is presented and described in detail. Instead of including all the produced code as appendices, as that would be impractical and decrease readability, it is chosen to include the most vital code blocks when needed throughout this chapter. Two different approaches are used; for simple code performing a specific task, or describing an isolated procedure, the original code from the source files is imported directly. The source file is then indicated in the caption and the line numbers corresponds to the line numbers in the original file. All source files can be found in the archive supplied together with the report. More advanced procedures that either requires a large amount of code, or algorithms that have their functionality spread across multiple functions, will instead be described using a simplified, but functionally equivalent, pseudo code. Here as well the source file containing the actual implementation is specified in the caption, but no line numbers are given.

3.1. Implementation and debug situation

During the whole design and implementation period the author was located at NTNU in Trondheim, meaning that there was no immediate access to the representatives of Hatteland Computer except through e-mail and telephone. A set of hardware modules was shipped to Trondheim to allow the implementation to be tested and debugged. This kit contained two AP's, four nodes and one POE power supply to power the AP's. This enabled testing of most of the features of the protocol, but not all. For instance there was no synchronization signal present, making that part of the code virtually un-testable. This also meant that there was no effective way to test functionality connected to the use of more than one channel, since the AP's, even if there were two available, would not be synchronized. Lastly the number of nodes was limited, and even with one AP acting as a node the maximum number of competing nodes would be 5. This meant that any artifacts appearing with a higher number of nodes would not be detected during the debugging. As seen in section 6.1 this did indeed cause a variety of problems throughout the test period

3.2. Implementation tool

During implementation the tool CodeVisionAVR V2.60 [6] was used with its default compiler and compiler settings. A lisence was supplied by Hatteland Computer for the full version. This tool was also used for programming the modules, with the exception of when debugging was to be done with JTAG, in these cases AtmelStudio 6.1 [7] was used and the compiled object file from CodeVisionAVR was imported into AtmelStudio.

Code 3.1: Convenience functions for transceiver access [ASRadioGeneral/general_radio.h].

```
void GR_generalRadioInit();

//------ Checksum Calculation -----------------------------------------------
//Calculates Checksum directly from the framebuffer
uint16_t GR_calculateChecksum(uint8_t start, uint8_t end);
uint8_t GR_validChecksum(uint8_t length);

inline uint16_t GR_get16BitAt(uint8_t pos);

#define GR_initiateCCA()      (PHY_CC_CCA |= (1 << CCA_REQUEST))
#define GR_channelIdle()      (TRX_STATUS & (1 << CCA_STATUS))
#define GR_disableReception()   (RX_SYN |= (1 << RX_PDT_DIS))
#define GR_enableReception()    (RX_SYN &= ~(1 << RX_PDT_DIS))

inline void GR_activateCCA_ED_DONE_I();
#define GR_deactivateCCA_ED_DONE_I() (IRQ_MASK &= ~(1 << CCA_ED_DONE_EN))

inline void GR_activateRX_END_I();
#define GR_deactivateRX_END_I()(IRQ_MASK &= ~(1 << RX_END_EN))

#define GR_getTRX_STATUS()   (TRX_STATUS & TRX_STATE_MASK)
#define GR_getTRAC_STATUS() ((TRX_STATE & TRAC_STATUS_MASK) >> 5)
#define GR_setTRX_CMD(cmd)   (TRX_STATE = cmd & TRX_CMD_MASK)
#define GR_fcsValid()      (PHY_RSSI & (1 << RX_CRC_VALID))
```

Code 3.2: IEEE 802.15.4 PHR type [ASRadioGeneral/general_radio.h].

```
typedef struct __PHR{  //PHY header
  union{
    struct{
      unsigned int length : 7; // Frame length
      unsigned int IEEE   : 1; // Frame IEEE compliant or not
    };
    uint8_t byte;
  };
} __PHR_t;
```

3.3. Module independent code

3.3.1. Transceiver control

In order to simplify the programming a simple interface is made to access the most commonly used parts of the transceiver. This interface was expanded gradually as the need for new functionality became clear.

The interface gives intuitive defines for activating and deactivating interrupts, checking and setting the transceiver state etc. In code 3.1 and 3.2 parts of this is shown to give a general idea of the approach.

3.3.2. Symbol counter

With the symbol counter being the core timing module of the protocol it is vital to have a simple and easy to use interface for this as well. In code 3.3 and 3.4 the interface for one counter is presented, the other two is exactly similar, only with other registers being accessed.

Code 3.3: Symbol counter headers [ASRadioGeneral/mac_symbol_counter.h].

```c
inline void MSC_initMacSymbolCounter();
#define MSC_forceBeaconTimestamp() (SCCR0 |= (1 << SCMBTS))

inline void MSC_setCompare1(uint32_t ticks);
inline void MSC_setCompare116Bit(uint16_t ticks);
inline void MSC_reactivateCompare1();
#define MSC_deactivateCompare1() (SCIRQM &= ~(0x01))
#define MSC_compare1Set() (SCIRQS & 0x01)
```

Code 3.4: Symbol counter implementations [ASRadioGeneral/mac_symbol_counter.c].

```c
inline void MSC_setCompare1(uint32_t ticks){
  SCOCR1HH = ticks >> 24;
  SCOCR1HL = ticks >> 16;
  SCOCR1LH = ticks >> 8;
  SCOCR1LL = ticks;
  //Clear any pending interrupts
  SCIRQS = 0x01;
  //Activate interrupt for given compare
  SCIRQM |= 0x01;
}

inline void MSC_setCompare116Bit(uint16_t ticks){
  SCOCR1LH = ticks >> 8;
  SCOCR1LL = ticks;
  //Clear any pending interrupts
  SCIRQS = 0x01;
  //Activate interrupt for given compare
  SCIRQM |= 0x01;
}
```

3.3.3. Symbol counter busy-wait function

In order to ensure accurate timing, specifically when interpreting a frame at the same time as it is being received, some dynamic delay functionality is needed. To accomplish this two functions are implemented in the symbol counter driver, one to initialize the procedure, and one to perform a busy wait loop until the counter has reached, or passed, the desired value. The two functions can be seen in code 3.5. This enables accurate synchronization of the frame buffer reading when needed, and if parts of the data interpretation require a lot of calculations, causing the microcontroller to lag behind the transceiver, the functions will simply not perform any waiting for the following calls. This makes the microcontroller drift back in sync without ever being in the risk of reading a byte too early, so long as the wait loop is initialized and used correctly.

3.3.4. Protocol definitions

As well as the functionality for accessing the hardware the protocol definitions are also common for all modules throughout the system. These are included in the general radio interface shared by the separate projects implementing the different modules. In code 3.6 type definitions for the headers and frame structures are presented. In addition to this the calculation of the custom checksum is presented in code 3.7.

Code 3.5: Busy wait functionality [ASRadioGeneral/mac_symbol_counter.c].

```c
static uint8_t Reference;
inline void MSC_setReference(){
  while(SCSR & (1 << SCBSY))delay_us(1);
  Reference = SCCNTLL;
};
inline void MSC_waitXAndSetReference(uint8_t x){
  uint8_t tmp = 0;
  do{
    while(SCSR & (1 << SCBSY))delay_us(1);
    tmp = SCCNTLL - Reference;
  }while(tmp < x);
  Reference += x;
};
```

Code 3.6: Type definitions for core protocol structures [ASRadioGeneral/general_radio.h].

```c
typedef struct __FRAME_HEADER{
  union{
    struct{
      unsigned int reserved   : 2;
      unsigned int networkId  : 4; // Network/System ID
      unsigned int msgType  : 2; // 0 = beacon, 1 = ack, 2 = data
    };
    uint8_t byte;
  };
}__FRAME_HEADER_t;

typedef struct __BEACON_CTRL{
  union{
    struct{
      unsigned int msgNum   : 2; // Number of commands
      unsigned int ackNum   : 5; // Number of Acknowledges
      unsigned int reserved   : 1;
    };
    uint8_t byte;
  };
}__BEACON_CTRL_t;

typedef struct __BEACON_HEADER{
  __PHR_t phr;
  __FRAME_HEADER_t frameHeader;
  __BEACON_CTRL_t beaconCtrl;
}__BEACON_HEADER_t;

typedef struct __DATA_HEADER{
  __PHR_t phr;
  __FRAME_HEADER_t frameHeader;
  uint16_t node;
}__DATA_HEADER_t;
```

Code 3.7: Checksum function [ASRadioGeneral/general_radio.c].

```
uint16_t GR_calculateChecksum(uint8_t start, uint8_t end){
  uint8_t i = start;
  uint8_t sum1 = 0;
  uint8_t sum2 = 0;

  if(end > 127) end = 127;

  //Checksum is xor of all bytes.
  while(i < end){
    sum1 ^= (&TRXFBST)[i++];//First byte -> even numbered (0,2,4..)
    sum2 ^= (&TRXFBST)[i++];//Second byte -> odd numbered (1,3,5..)
  }
  //Special case with only one byte, second byte is 0.
  if(start == end) sum1 ^= (&TRXFBST)[start];

  return (sum1 << 8) | sum2;
}
```

3.4. Interface design

As with most software development it is desirable to implement the protocol as a standalone module that a user application can access through a simple interface. This will both simplify the development and debugging as well as clearly defining the boundaries of the work to be done in this thesis. All application specific features are abstracted away and only the core functionality is of any interest.

Given the deep transceiver to software integration atmega128RFA1 offers, through interrupts and direct register access, it is decided to make the protocol completely interrupt driven. By using the mac symbol counter as timing reference it will be possible to keep all radio functionality hidden from the user. Once the application has initiated the protocol and enabled interrupts no further protocol actions has to be made within the main loop besides accessing the buffers.

Still the application has to be designed to allow the protocol to do its job as effectively as possible. This means that the use of interrupts besides the ones used by the protocol must be minimized. This to prevent that the protocol interrupts are delayed because of other ongoing interrupts. This will easily lead to stability issues and, in the worst case, cause the protocol to fall apart if the beacons are thrown completely out of sync.

3.4.1. Identification and addressing

The nodes will have a unique node ID set by the application. It is the responsibility of the application to ensure the uniqueness, and an address conflict will cause unpredictable behavior. The protocol will initialize the node ID to the highest available value, FFFF in hexadecimal, and the application must set the new ID using the supplied interface functions after initializing the protocol. A suggested procedure for assigning node ID's are presented in section 4.2.4.

The network ID should also be set by the application, both on the node and on the AP side. It shall be initialized to the highest available value, F in hexadecimal, on the AP side, while it will be initialized to 0 on the nodes. 0 indicates a special mode where the module will accept all network ID's, a frame with network ID 0 will also be accepted by AP’s with a different ID, meaning that a node without a specified network ID will synchronize and communicate with any AP in range. Note that this feature has not been implemented as it is of no significance for the core functionality of the protocol. As such this functionality should be added at a later stage.

3.4.2. Buffers

The interface is made up of two buffers, one for output and one for input. These buffers consist of a specialized structure containing all the information connected to a message, including a collection of flags used to indicate the status of the transmission. The structures are presented in code 3.8.

The same structures are used for both the input and output buffers on both the nodes and the AP’s, this is done to keep the interface unified and easier to use for the application. Still this means that the out buffer in the AP is able to hold messages that are far larger than what the beacon are able to fit. This means that the application has to ensure that messages added to the buffer is within the limits. The maximum message size will be defined in the interface. If a message is found to be too large for a single transmission the application can split it into several smaller pieces.

The node ID will have a different meaning depending on the direction of the message. This comes from the nature of the protocol and the fact that the AP’s are not identified in any way. For messages being transmitted from the nodes to the AP’s the node ID identifies the sender node, while for messages being transmitted from the AP’s to the nodes the node ID indicate the receiving node. For the node the protocol will overwrite any node ID with the node ID of the node, making it impossible for the application to fake its ID. When receiving messages the protocol will filter the messages based on the node ID, only letting through directly addressed messages and broadcasted messages with node ID 0. The control flags are only used in the output buffer, as the reception of frames does not require any control logic besides the indexes. Still the control byte is included to enable the use of the same structures across the whole system.

3.4.3. Possible race conditions

One important thing to consider in this situation is the fact that the indexes can be modified by two independent and concurrent sources. This is a scenario where race conditions can easily cause undefined behavior. The application can be interrupted by the protocol while being in the middle of updating an index, causing the protocol to read the old value instead of the new value. If the protocol is planning on updating the same index the end result will be that the new value written by the protocol is immediately overwritten by the application when the interrupt routine returns.

The obvious solution is to make the macros atomic. However when going through the functionality of the interface it is clear that each index will only ever be modified by one of the two interacting parts, while the other part will only read the value.

The protocol will increment the In_Begin index when a new message is received, the application on the other hand will never modify this index, only read it. In_End on the other hand is only modified by the application, once it is ready to release that buffer slot.

Code 3.8: General interface types and functions [ASRadioGeneral/general_radio_if.h].

```c
//------ Messages -------------------------------------
#define MAX_MESSAGE_DATA_SIZE 119
typedef struct __MSG_CTRL{
  union{
    struct{
      unsigned int sent    : 1; // Message sent
      unsigned int sending : 1; // Message being sent
      unsigned int acked   : 1; // Message acknowledged
      unsigned int reject  : 1; // Message rejected
      unsigned int toBig   : 1; // Message to large
      unsigned int       : 3;
    };
    uint8_t byte; //Should only be used to clear all flags
  };
}__MSG_CTRL_t;

typedef struct __MESSAGE{
  uint16_t node;    // Node ID
  uint8_t length;   // Length of data
  uint8_t data[MAX_MESSAGE_DATA_SIZE]; // Data
}__MESSAGE_t;

typedef struct __MESSAGE_OBJ{
   __MSG_CTRL_t ctrl;
   __MESSAGE_t msg;
}__MESSAGE_OBJ_t;

//-------- IN BUFFER --------------------------------------
volatile extern __MESSAGE_OBJ_t In_Buffer[IN_MAX+1];
volatile extern uint8_t In_Begin, In_End;
#define incrementInIndex(x) (x = (x+1)%(IN_MAX+1))
#define inBufferEmpty() (In_End == In_Begin)
#define inBufferFull() ((In_End == In_Begin-1) || (!In_Begin && In_End == IN_MAX))

//-------- OUT BUFFER --------------------------------------
volatile extern __MESSAGE_OBJ_t Out_Buffer[OUT_MAX+1];
volatile extern uint8_t Out_Begin, Out_End;
#define incrementOutIndex(x) (x = (x+1)%(OUT_MAX+1))
#define outBufferEmpty() (Out_End == Out_Begin)
#define outBufferFull() ((Out_End == Out_Begin-1)||(!Out_Begin && Out_End == OUT_MAX))
```

The out buffer indexes will only be modified by the application because of the control flags, Out_Begin is incremented when a new message is added to the buffer, the protocol detects the index change and transmits the message, setting the control flags according to the status of the transmission, once the message is transmitted, successfully or not, it is up to the application to increment the Out_End index to make room for more messages.

The protocol will use internal indexes, not visible to the application, to keep track of what messages has been sent and acknowledged. This way no race conditions will ever occur and the need for atomic operations is not present.

Code 3.9: AP specific interface [ASRadioAP/ap_radio_if.h].

```c
//-------- BUFFERS ----------------------------------------
#define IN_MAX 40
#define OUT_MAX 10

//---- Include Buffer definitions ----
#include "../ASRadioGeneral/general_radio_if.h"

//-------- Functions --------------------------------------
//Initializer
void initApRadio();

//NetId initialized to 0
void setNetId(uint8_t netId);

//Synchronization function, to be called at exact 1Hz
void globalSecTick();

//Overrides the internal channel analysis
void forceChannelSwap();
```

Code 3.10: Node specific interface [ASRadioNode/node_radio_if.h].

```c
//-------- BUFFERS -------------------------------------
#define IN_MAX 5
#define OUT_MAX 3

//---- Include Buffer definitions ----
#include "../ASRadioGeneral/general_radio_if.h"

//-------- Functions -----------------------------------
void initNodeRadio();

//Node initialized to nodeNo 0xFFFF.
void setNodeNo(uint16_t no);

//NetID initialized to 0
void setNetId(uint8_t netId);
void blackListNetId(uint8_t netId);

//Returns 1 if node is synced to one or more AP's. 0 if not.
uint8_t connected();
```

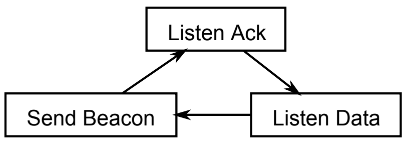


Figure 3.1.: Simple state machines for the access point module.

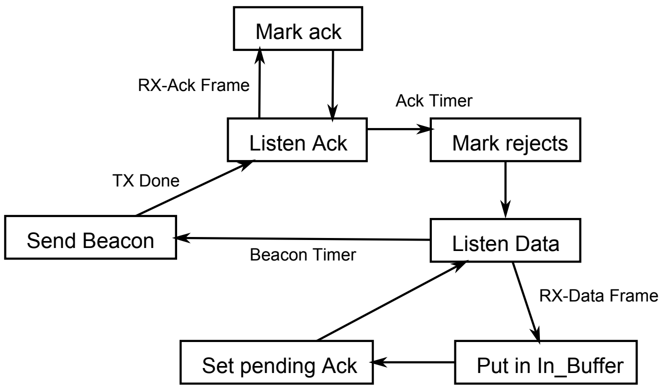


Figure 3.2.: Expanded state machine for the access point module.

3.5. AP protocol core

3.5.1. State machine

In isolation the behavior of a single AP, and thus a single channel, is simply the transmission of a beacon, followed by a dedicated acknowledge phase, followed by an access window where anyone can transmit using CSMA/CA. A simple state machine for this is shown in figure 3.1.

In figure 3.2 the state machine from figure 3.1 are expanded with the major sub states and the triggers causing the transitions. As seen in this figure the main triggers are the reception of a frame, either acknowledge or data, the completion of beacon transmission, and two timers, one indicating that the acknowledge phase is over and the other indicating that it is time to transmit a new beacon.

3.5.2. Timers

The two main timers are handled by the MAC symbol counter. The values of the timers will depend on the configuration of the system, most importantly the beacon frequency. The configuration options and calculations are presented in section 3.9.

The beacon timer will trigger each time the AP is to transmit a beacon. The timer will be set according to the current channel and the global timing reference.

The acknowledge timer will be set to trigger two subperiods after the beacon trigger. One subperiod is again equal to the beacon period divided by the number of available channels. When this timer triggers the AP will look through the acknowledgements received during that phase and match them with the messages that were included in the beacon. Any message that does not have a matching acknowledgement will be marked as rejected while the rest is marked as acknowledged.

3.5.3. Frame reception triggers

The reception of a frame is indicated with two interrupts. First the RX_START interrupt that triggers once a correct preamble, SFD and PHR is received. This interrupt enables the module to read the incoming frame while it is being received. With this approach it has to be made sure that the bytes are not read faster than the radio is receiving and writing them. In order to ensure this the symbol counter busy wait function described in section 3.3.3 can be used. Another aspect is that it is impossible to know if the data is corrupted before the full frame is received and the checksums are calculated. This means that it must be possible to invalidate any actions or values set throughout the interpretation.

The second interrupt is RX_END, this triggers when a full frame has been received and also indicates that the automatic FCS is done calculating. If the reading and interpretation of the frame is started by this trigger, the validity of the frame can be tested before starting the read procedure. Thus the fallback feature needed when using RX_START is no longer needed. Also no concern has to be made regarding delays since all the data is ready in the buffer to begin with. The use of this interrupt introduces another pitfall however; unless reception of new frames is disabled, the radio will overwrite the buffer when a new frame is received. This means that the routine reading the buffer has to ensure that no byte is read at a time where is can possibly have been overwritten. The minimum theoretical timing between two frames "on-air" is $140\mu s$ for the CCA [3] plus 10 symbols $(160\mu s)$ for the preamble and the SFD. The last byte of the physical header is the PHR containing the length of the frame, and this is the first byte accessible to software that will be overwritten. After this the data bytes of the frame will be overwritten one by one every 2 symbols.

In practice it is possible to read the previous frame while receiving a new one so long as data byte i is not read later than $(332+i*32)\mu s$ after the RX_END interrupt. In most cases this should be a relatively simple requirement to satisfy regarding the data, with the exception of the frame length byte that will be needed to know when to stop reading. To prevent corruption of this value it should be copied into a temporary variable instead of using the register directly.

Given the above considerations there is little gain for the AP in reading the data frames in parallel, meaning that the simplest solution with reading once the frame is completed is chosen. The interpretation of the received frames are shown in code 3.11, the functions encoded with ACM_* are related to the channel management functionality and is further described in section 3.8. The function for reading the data frame is shown in code 3.12.

3.5.4. Beacon building

In the early versions of the protocol there were significant stability issues. Nodes would disconnect randomly and the overall performance showed that a large amount of frames were lost. After some analyzing this was narrowed down to the accuracy of the beacon timing. To begin with the whole process of building the beacon and filling the frame buffer was done when the beacon timer triggered. Since the computation time needed to build the frame would vary depending on the content, the actual transmission of

Code 3.11: Frame reception interrupt [ASRadioAP/ap_radio.c].

```c
interrupt [TRX24_RX_END] void receptionCompleted(){
  uint8_t length = TST_RX_LENGTH;

  //Check validity of frame
  if((Flag.IncludesChecksum && !GR_validChecksum(length)) || !GR_fcsValid()){
    ACM_registerFailedFrame();
    return;
  }

  //Read contents
  if(Incomming_Frame_Header.networkId == NetId){
    switch(Incomming_Frame_Header.msgType){
      case BEACON_FRAME:
        ACM_registerOtherNetwork();
        break;
      case DATA_FRAME:
        frameInterrupt();
        if(Incomming_Frame_Header.reserved) ACM_readChInfoFromFrameBuffer(length-8);
        break;
      case ACK_FRAME:
        Inc_Ack[Inc_Ack_End] = GR_get16BitAt(1);
        Inc_Ack_End++;
        break;
      case BROADCAST_FRAME:
        break;
    }
  }else{
    ACM_registerOtherNetwork();
  }
}
```

Code 3.12: Data frame read function [ASRadioAP/ap_radio.c].

```c
void frameInterrupt(){
  uint8_t i = 0;
  //If receive buffer is full, discard message
  if(inBufferFull() || Ack_End > 31) return;

  //Copy message into receive buffer
  In_Buffer[In_End].msg.length = (&TRXFBST)[3];
  In_Buffer[In_End].msg.node = GR_get16BitAt(1);
  for(i = 0; i < In_Buffer[In_End].msg.length; i++)
    In_Buffer[In_End].msg.data[i] = (&TRXFBST)[4+i];

  //Append node ID to the Ack list
  Ack_Pending[Ack_End] = In_Buffer[In_End].msg.node;

  //Update indexes
  Ack_End++;
  incrementInIndex(In_End);
}
```

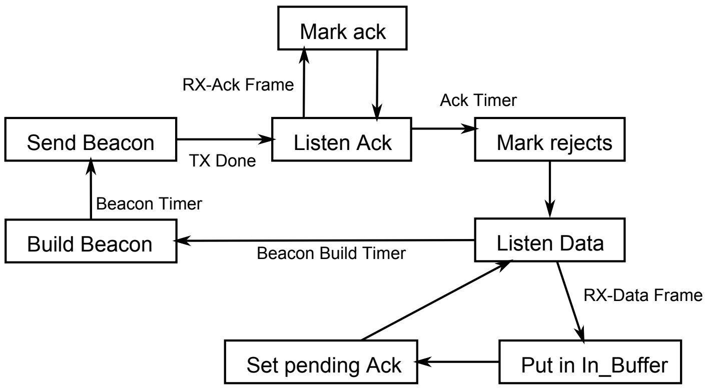


Figure 3.3.: AP state machine with the addition of the beacon build

the beacon could be delayed by an unpredictable amount of symbols. This caused the nodes to push their beacon timer forward, expecting that the change was legitimate. If a computationally heavy beacon was followed by a nearly empty one there was a risk that the second transmission was initiated before the nodes had locked on to the channel, causing the nodes to miss the preamble and thus miss the entire frame. The solution to this was to introduce a third timer initiating the beacon building. This timer will trigger a set period before the beacon timer and prepare the beacon frame in the frame buffer. When the beacon timer triggers its only task will be to initiate the transmission, giving a high degree of timing accuracy. The further expanded state machine for the AP module including this timer is seen in figure 3.3. Once this was in place the stability of the protocol increased dramatically.

The building of the beacon consists of three main phases. The first phase involves inserting all the acknowledgements that has been generated during the last access window. Given the definitions of the protocol the acknowledgements should have priority over the messages, meaning that as many acknowledgements as possible should be inserted up to the maximum 31. The timing of this phase will be proportional to the number of acknowledgements plus a constant factor. Each acknowledgement takes up two bytes, as it is simply the id of the transmitting node.

The second phase is the addition of messages. Firstly it has to be checked that there are pending messages and that the first message in line is not too large to fit within the remaining free space of the beacon. Secondly the protocol states that each node can only receive a single message per beacon as discussed in section 2.5.5. To ensure this limit some extra structures are needed, specifically it is needed to filter the addresses to see if the next pending message has the same address as another, already inserted, message. Secondly a broadcast message can only be inserted if it is the first message in the queue, and after the insertion of such a message no more messages can be added even if there is still room for more.

The third phase is the construction and insertion of the header information, including the PHR and the checksum described in section 3.3.4. Pseudo code for the beacon

Code 3.13: Pseudo code for the beacon building procedure [ASRadioAP/ap_radio.c].

```
prepareBeaconFrame(){

//--- Phase 1 - Acknowledge insertion -----
  if(NormalBeacon){ //First beacon on a new channel shall be empty
    while(moreAcksLeft AND moreRoomInBeacon){
      InsertNextAckInFrameBuffer();
    }
    SetAckNumberInBeaconHeader(AckNumber);
  }
  //Acks that did not fit in frame are discarded as nodes will only listen once.
  ResetOutGoingAckBuffer();

//--- Phase 2 - Message insertion -----
  if(NormalBeacon){
    //Append as many messages as possible without exceeding max size.
    while(PendingMessage AND RoomForMessageInBeacon){
      //Prevent more than one message per node per beacon
      if(MessageIsBroadcast AND OtherMessagesAlreadyAddedToBeacon) break;
      if(MessageAddressAlreadyPresentInBeacon) break;
      AddMessageAddressToAddressMask(Message.NodeID);
      InsertCompleteMessageIntoFrameBuffer();
      IncrementMessageNumberInBeaconHeader();
      //Allow only a single message if the message is a broadcast
      if(MessageIsBroadcast) break;
    }
  }

//--- Phase 3 - Header and checksum insertion -----
  InsertFrameHeaderInfo();
  CalculateAndInsertCustomChecksum();
}
```

building helper function can be seen in code 3.13. The different phases are separated with comment lines.

3.5.5. Acknowledge reception

Once a beacon has been transmitted the AP will enter acknowledge mode for the duration of the next subperiod. More accurately it will enter this mode immediately after having transmitted the beacon, and stay there until the following subperiod is completed. This will happen regardless of whether there were messages in the beacon or not since the acknowledge phase is blocked from being used for normal data traffic in any case. The state transition is triggered by the TRX24_TX_END interrupt that triggers immediately after the last frame symbol has been transmitted on air.

The acknowledge phase is split into three slots where the nodes transmit their acknowledge frames according to what index their message had in the beacon. This means that the third slot will only be used when there are three messages in the beacon etc. The received acknowledgements are temporarily stored in an array for the duration of the acknowledge phase. Once the timer indicating the transition to the access window triggers, the AP will loop through the received acknowledgements marking the transmitted messages as rejected or acknowledged accordingly. This routine is shown in code 3.14. This routine also contains code for changing channels, this part is further described in section 3.5.6.

The transmission of broadcast messages are naturally not acknowledged, still the ac-

Code 3.14: Acknowlede phase completion routine [ASRadioAP/ap_radio.c].

```c
interrupt [SCNT_CMP2] void ackPeriodCompleted(){
  uint8_t a = 0;
  uint32_t offset;
  MSC_reactivateCompare2();

  State = LISTEN;
  //Loop through received ack's and the messages waiting for ack. Mark as needed.
  a = 0;
  while(Out_Unacked != Out_Unsent){
    if(  a < Inc_Ack_End
      && Out_Buffer[Out_Unacked].msg.node == Inc_Ack[a])
    {
      Out_Buffer[Out_Unacked].ctrl.acked = 1;
      a++;
    }else if(!Out_Buffer[Out_Unacked].msg.node){
      //Broadcasts are always marked as acked.
      Out_Buffer[Out_Unacked].ctrl.acked = 1;
    }else{
      Out_Buffer[Out_Unacked].ctrl.reject = 1;
    }
    incrementOutIndex(Out_Unacked);
  }
  Inc_Ack_End = 0;

  if(Flag.ChChange){
    a = (Channel - OldChannel);
    while(a >= CH_LAST) a -= CH_LAST;
    a -= CH_FIRST - 1;
    if(a < 4) a += CM_ChCount();

    offset = (uint32_t)a * SUBPERIOD;
    MSC_setCompare1(offset);
    MSC_setCompare3(offset - BEACON_BUILD_TIME);

    CM_apply_channel(Channel);
    Flag.Synced = 1;
    Flag.ChChange = 0;
    Flag.EmptyBeacon = 1;
    Flag.RssiReading = 0;
  }
}
```

knowledge status is set for any transmitted broadcasts at the end of the acknowledge phase to signal that the message has been transmitted. This is done to simplify the interface, allowing the application to always listen for the acknowledge/reject statuses to determine when a message can be cleared from the buffer.

3.5.6. Changing channel

Upon request from the application, or upon the channel quality falling below the desired level, the AP will move to a different channel. The evaluation and ranking of the channels, as well as how the information is gathered, is presented in section 3.8 as this can be seen as a standalone feature besides the normal operation of the protocol.

The actual channel change however has to be incorporated into the protocol to make sure it does not cause unnecessary problems. First of all it is not indifferent at what point in the beacon sequence the channel swap is executed. The subperiod where the AP is actually transmitting its beacon is obviously excluded, and this extends to the acknowledge phase. This way it can be ensured that the messages transmitted by the AP do not suffer unnecessary from the channel change. The received messages however are bound to trigger retransmissions. If the AP listens for frames before it moves, the frames received will be falsely marked as unacknowledged by the transmitting node since it will be listening for the acknowledgement on the wrong channel. As such it is better that the AP swap channels just as the data phase is starting, this way the data frames transmitted within the data period is truly lost, giving a more consistent and correct view of the situation for all involved parties.

This means that the channel change should be executed upon the completion of the acknowledge phase. This can be seen in code 3.14.

To change channel firstly involves a change of transmission frequency, but given the functionality of the protocol it also involves that the AP has to recalculate its beacon timing to adjust to its new position in the beacon sequence. This is done by calculating the offset between the current channel and the new, and then set the timers according to the new channel. If the offset is very small, meaning that the new channel has a beacon slot within only a few subperiods following the original channel, a full period is added to the offset to make sure that the AP has time to initiate the beacon transmission properly.

Code 3.15: Pseudo code for first sync. function [ASRadioAP/ap_radio_.c].

```
globalSecTick(){
  SymbCnt = GetSymbolCounter();
  if(ExpectedSymbCnt > SymbCnt){
    SymbolDiff = ExpectedSymbCnt - SymbCnt;
    ExpectedSymbCnt += 62500 - CorrectionFactor;
  }else if(ExpectedSymbCnt < SymbCnt){
    SymbolDiff = SymbCnt - ExpectedSymbCnt;
    ExpectedSymbCnt += 62500 + CorrectionFactor;
  }else{
    SymbolDiff = 0;
    ExpectedSymbCnt += 62500;
  }

  if(SymbolDiff < ErrorLimit){
    Offset = (CurrentChannel * SubperiodTiming) + Adjustments;
    SetTimers(Offset);
  }
}
```

3.6. AP synchronization

In order to keep the access points synchronized a global timing signal is supplied. This is defined to be 1Hz in section 2.5.8, and should reset the timers of the module to the correct values according to the currently active channel. This will make sure that all access points are at their dedicated position within the channel cycling sequence.

One main thing to take into consideration when designing this function is that the triggering of the interface function can be delayed by ongoing interrupts. This means that the timers cannot be set directly, but only when it can be assumed that the signal is not delayed.

3.6.1. Initial version

The initial approach to this was to use the symbol counter as a reference, setting an "expected symbol count" for when the synchronization should arrive, and then only trigger the timers if the actual signal is within a given offset of the expected value. In addition to this a factor is added or subtracted to account for any clock offsets compared to 1Hz. Pseudo code for this procedure is presented in code 3.15.

During implementation the synchronization had to be tested using internal timers in the AP, as the global signal was not available. It would have been possible to supply a 1Hz signal from some outside source, using a separate microcontroller and some directly soldered wires, but this was not considered worthwhile at the time. The result of this decision was that the system seemingly worked perfectly, since any clock drift and other errors was not present when generating the sync signal locally on the AP chip.

3.6.2. Second version

Once a proper external sync signal was supplied the original functionality turned out to be more or less unusable, as described in section 6.1.3. The reason for the instability was that the static correction factor on the expected symbol count initially was too

Code 3.16: Pseudo code for second sync. function [ASRadioAP/ap_radio_.c].

```
globalSecTick(){
  SymbCnt = GetSymbolCounter();
  if(ExpectedSymbCnt > SymbCnt){
    SymbolDiff = ExpectedSymbCnt - SymbCnt;
    if(SymbolDiffIncreasing())DynamicOffset--;
    else(random1in4chance())DynamicOffset--;
  }else if(ExpectedSymbCnt < SymbCnt){
    SymbolDiff = SymbCnt - ExpectedSymbCnt;
    if(SymbolDiffIncreasing())DynamicOffset++;
    else(random1in4chance())DynamicOffset++;
  }else{
    SymbolDiff = 0;
  }
  ExpectedSymbCnt += 62500 + DynamicOffset;

  if(SymbolDiff < ErrorLimit){
    Offset = (CurrentChannel * SubperiodTiming) + Adjustments;
    SetTimers(Offset);
  }
}
```

small, causing the sync signal to drift away as the expected value was not incremented enough each time. This revealed the need for a dynamic factor, adjusted according to the current offset between the expected and the received value. Since this error was first discovered during the testing phase the general situation was rather stressful and the resulting function was mostly produced throughout trial and error rather than through a proper design process. Pseudo code for the second version is presented in code 3.16. The fundamental concept is simply to adjust the correctional factor if the symbol difference is growing. Otherwise the factor has a random chance of being modified so long as the difference is not zero.

Unfortunately this new version did not fully fix the issue, even if it provided a far more versatile solution that the initial function. However the functionality was not improved further for other reasons described in section 6.1.4.

A thorough analysis of the synchronization functionality is given in section 8.2 with proposals for future improvements.

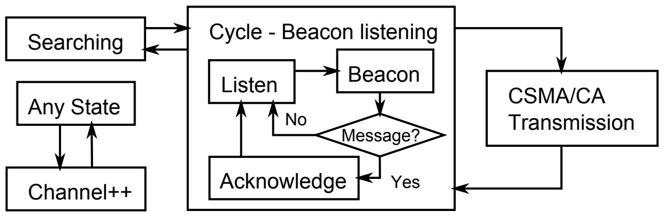


Figure 3.4.: Simple state machine for the node modules.

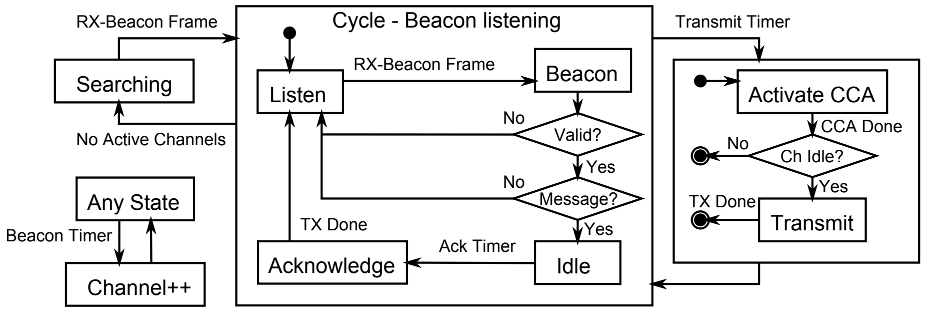


Figure 3.5.: Expanded state machine for the node modules.

3.7. Node protocol core

3.7.1. State machine

The main feature of the nodes is the cycling through all available channels. To begin with a search procedure has to be executed to locate and lock on to the global timing of the AP's. Once the node is synchronized it will follow the timing of the beacons, cycling through all channels accordingly. This sequence will only be interrupted when the node wish to send, at which point it will find a suited channel and attempt to access it using a customized CSMA/CA algorithm. A simple state machine for the node can be seen in figure 3.4. Note how the beacon timer and the corresponding channel incrementation is always performed no matter what state the node is in, this is to ensure that it can jump right back into the sequence after a completed transmission no matter how long the CSMA/CA and transmission procedure takes.

In figure 3.5 the simple state machine from figure 3.4 is expanded in greater detail and with the major triggers.

3.7.2. Timers

The functionality of the node is centered on the beacon timer, synchronized to the global timing each time a new beacon is received. When a beacon is analyzed, and a message addressed to that particular node is detected, the acknowledge timer is activated based on the index of the message. This cause the node to transmit its acknowledge within its designated slot in the acknowledge phase. The last timer is the transmit trigger, preparing the data frame and initiating the CCA. This timer is activated a random number of subperiods following the detection of a new message in the outgoing buffer. The timer itself also has a random value as it is to trigger at an arbitrary point in time within the given subperiod. This high degree of randomness is a fundamental part of the CSMA/CA procedure as it will minimize the probability that two nodes access the channel at the exact same time.

3.7.3. Searching

The search phase is very simple, the node will cycle through the channels as normal, but instead of listening on each channel for one subperiod's duration it listens for a full beacon period. This means that if there is an AP transmitting on the given channel the node is bound to pick up the beacon at some point during that period. Once a beacon is registered the global timing is immediately known and the node can commence with the normal cycling, detecting any beacons on other channels as it moves through the normal channel sequence.

If at any point during normal operation the node believes that there are no active channels it will go back to the search phase until it once again receives a beacon.

With 16 channels and a beacon frequency of around 30Hz it will take the node a maximum of just over 0.5 seconds to find and lock on to the beacons. This also means that it will take the node 0.5 seconds to confirm that no channels are active if all AP's go offline. This is particularly interesting in connection to the emergency stop function mentioned in section 1.2.

3.7.4. Beacon reception

After locking onto a new channel in the channel sequence, the node will stay idle, listening. When a valid preamble and SFD is detected the TRX24_RX_START interrupt indicates that a frame is being received. In order to maximize the throughput of the protocol it is essential to minimize the portion of the subperiod that is not actively used for beacon transmission, as described in section 2.5.6. Because of this it is desired to do the majority of the needed computation in parallel. Still, as stated in section 3.5.3, this requires the possibility to undo any changes invoked by the interpretation in case the frame is corrupted.

For the beacon reception this is relatively easy to accomplish by the use of flags. One flag indicates if an acknowledgement targeted at the specific node was detected, if the beacon frame is validated this flag is used to mark the transmitted frame as acknowledged or rejected. If the beacon frame was invalid the transmitted frame has to be marked as rejected regardless of the flag value. Any messages targeted at the node, both addressed and broadcasts, can be copied into the incoming buffer during interpretation if the buffer is not full, a flag will then indicate if a message has been copied and a variable indicates the index of the message in the beacon message list. The fallback procedure is then simply to leave the index of the in buffer unchanged if the frame was invalid, thus the copied data is simply overwritten the next time a message is received. A valid frame will cause the index to be incremented normally and the acknowledge timer to be activated according to the stored message index if the message was directly addressed.

Initially the beacon was interpreted after the full frame had been received. This was done in order to get the protocol up and running as fast as possible. The transition to parallel interpretation was then made at a later stage once the conceptual functionality was confirmed to work as intended.

In table 3.1 some test results from the debugging phase is presented, showing how the stability of the protocol increased drastically once this change was made. The presented data are the numbers accumulated since the module was last reset, meaning that the higher the numbers the longer the system has been operating. Initially the system worked well with 30 beacons per second, when this was increased to 35 the system got unusable with close to 25% lost packets and disconnects at a regular basis. With the change to parallel interpretation of the beacons the numbers changed drastically and the system was able to run for days at 35Hz with only a completely insignificant amount of unexpected errors.

3.7.5. Acknowledge transmission

The node transmits the acknowledge frame according to the index of the received message in the beacon frame, the setting of the timer can be seen in code 3.17. The interrupt routine of the acknowledge timer is shown in code 3.18. One thing to note is that the acknowledge frame is being built while it is being transmitted. The transmission is initiated immediately after inserting the PHR. Then the remaining data, the header and the node ID, is inserted while the transceiver is busy transmitting. This is possible because the transceiver is transmitting far slower than the microcontroller is writing. This allows the code related to the acknowledge frame to be kept in one place, while still ensuring that the transmission is initiated as close to the desired time as possible. The acknowledge frame does not add the custom checksum, as explained in section 2.5.4.

3.7.6. Active channel registration

Given the dynamic nature of the channels the nodes will pick up the beacons from all AP's within range. The channels are then marked as active from the node's point of view and can be used for transmission. When a node listens for a beacon on an active channel without receiving any, it has to assume that it has moved out of that AP's range or that the AP has gone offline. Given that the beacon frequency is sufficiently high compared to the maximum physical speed of the nodes it will be possible for the nodes to consider its list of active channels to be correct at all times.

When the node wishes to transmit, it will have to choose a channel for transmission. This could be done randomly among the active ones, but naturally it is better if the node choose the channel that appears to have the best signal strength, indicating that the physical distance is the shortest or that destructive interference is the lowest. The ATmega128RFA1 chip implements an energy measurement according to the 802.15.4 standard [5] to enable this, this is further described in section 2.3.2, and is a measurement of the energy of any received signal on the channel. By storing this value each time a beacon is received the node will have a good indicator on how strong the signal is on that particular channel, enabling the channels to be ranked in an effective way.

```
Test results 09:57 8. mars 2013: 30 Beacons/sec
Timings changed from 20 b/s to 30 b/s.

Node:  1    - Sent:   51704 - Recv:   50302 - (   51703 /    0)
 B:   74340 - Recv:   51703 - Sent:   50485 - (   50301 /  184) - DC’s:  0
Node:  2    - Sent:   51703 - Recv:   50010 - (   51702 /    0)
 B:   74340 - Recv:   51702 - Sent:   50469 - (   50009 /  460) - DC’s:  0
Node:  3    - Sent:   51703 - Recv:   50078 - (   51702 /    0)
 B:   74340 - Recv:   51703 - Sent:   50520 - (   50077 /  443) - DC’s:  0
Node:  4    - Sent:   51703 - Recv:   50407 - (   51702 /    0)
 B:   74339 - Recv:   51701 - Sent:   50414 - (   50406 /    8) - DC’s:  0
```

```
Test results 10:11 8. mars 2013: 35 Beacons/sec
Timings changed from 30 b/s to 35 b/s.

Node:  1    - Sent:    2791 - Recv:    2041 - (    2295 /  495)
 B:    3844 - Recv:    2326 - Sent:    2041 - (    2004 /   37) - DC’s: 36
Node:  2    - Sent:    2443 - Recv:    1670 - (    1815 /  627)
 B:    3511 - Recv:    1844 - Sent:    1681 - (    1623 /   58) - DC’s: 46
Node:  3    - Sent:    2666 - Recv:    1887 - (    2098 /  567)
 B:    3670 - Recv:    2131 - Sent:    1895 - (    1846 /   49) - DC’s: 40
Node:  4    - Sent:    2621 - Recv:    1841 - (    2057 /  563)
 B:    3673 - Recv:    2086 - Sent:    1856 - (    1799 /   57) - DC’s: 41
```

```
Test results 11:12 23. mars 2013: 35 Beacons/sec
Beacons are now interpreted while being received.

Node:  1    - Sent: 3687974 - Recv: 5560166 - ( 3685053 /  2920)
 B: 9024962 - Recv: 3687970 - Sent: 5564583 - ( 5560165 /  4418) - DC’s:  0
Node:  4    - Sent: 3687946 - Recv: 5548594 - ( 3681085 /  6860)
 B: 9024910 - Recv: 3687927 - Sent: 5565838 - ( 5548586 / 17252) - DC’s:  2
Node:  6    - Sent: 3687974 - Recv: 5494521 - ( 3670926 / 17047)
 B: 9024957 - Recv: 3687972 - Sent: 5579759 - ( 5494520 / 85239) - DC’s:  0
Node:  7    - Sent: 3687973 - Recv: 5514514 - ( 3675147 / 12825)
 B: 9024959 - Recv: 3687970 - Sent: 5581907 - ( 5514513 / 67394) - DC’s:  0
Node:  9    - Sent: 3678635 - Recv: 5504148 - ( 3668045 / 10592)
 B: 9007645 - Recv: 3678613 - Sent: 5568175 - ( 5504136 / 64039) - DC’s:  3
```

```
Test results 14:22 25. mars 2013: 35 Beacons/sec
Fixed a bug causing the nodes to transmit data within the acknowledge-phase.

Node:  1    - Sent: 2411199 - Recv: 4597175 - ( 2411194 /      4)
 B: 6200028 - Recv: 2411194 - Sent: 4637167 - ( 4597171 /  39996) - DC’s:  1
Node:  2    - Sent: 2411199 - Recv: 4566480 - ( 2411197 /      1)
 B: 6200037 - Recv: 2411196 - Sent: 4662923 - ( 4566479 /  96444) - DC’s:  0
Node:  3    - Sent: 2411198 - Recv: 4623580 - ( 2411197 /      0)
 B: 6200039 - Recv: 2411197 - Sent: 4637116 - ( 4623579 /  13537) - DC’s:  0
Node:  4    - Sent: 2411191 - Recv: 4594623 - ( 2411190 /      0)
 B: 6200037 - Recv: 2411189 - Sent: 4658072 - ( 4594622 /  63450) - DC’s:  0
Node:  5    - Sent: 2411188 - Recv: 4549124 - ( 2411186 /      1)
 B: 6200039 - Recv: 2411186 - Sent: 4649751 - ( 4549123 / 100628) - DC’s:  0
```

Table 3.1.: Test results during debugging showing the improvement of parallel interpretation of the beacon frame.

Code 3.17: Pseudo code for beacon reception [ASRadioNode/node_radio.c].

```
receiveBeacon(){
  SetTimingReference();
  WaitXSymbols(3); //Wait for first byte to be written (frame header).
  FrameHeader = FrameBuffer[0];
  if(FrameHeader.networkId == MyNetId){
    if(FrameHeader.msgType == BEACON){
      ForceBeaconTimeStamp();
      SetBeaconTimer(CONSTANTVALUE); //Synchronize

      //--- Ack reception --------
      WaitXSymbols(2); //Wait for beacon header
      BeaconHeader = FrameBuffer[1];
      if(WaitingForAck){
        while(AcksLeftInBeacon){
          WaitXSymbols(4); //Wait for the next Ack
          if(NextAck == MyId) ReceivedAck = TRUE;
        }
      }else WaitXSymbols(BeaconHeader.AckNumber * 4); //Wait for the whole ack list

      //--- Message reception --------
      while(MessagesLeftInBeacon){
        WaitXSymbols(4);//Wait for address
        if(NoMessageReceived AND GetNextMessageAddress == MyId){
          InBuffer.Id = MyId;
          WaitXSymbols(2); //Wait for lenght
          InBuffer.Length = GetMessageLength;
          while(MoreDataInMessage){
            WaitXSymbols(2); //Wait for one byte
            InBuffer.NextData = FrameBuffer.NextData;
          }
        }else if(NoMessageReceived AND MessageIsBroadcast){
          InBuffer.Id = 0;
          WaitXSymbols(2); //Wait for lenght
          InBuffer.Length = GetMessageLength();
          while(MoreDataInMessage()){
            WaitXSymbols(2); //Wait for one byte
            InBuffer.NextData = FrameBuffer.NextData;
          }
        }else{ //Uninteresting messages
          WaitXSymbols(2); // Wait for lenght
          WaitXSymbols(GetMessageLength() * 2); //Wait for all the data
        }
      }

      //--- Conclude the validity of the frame -------
      WaitXSymbols(4); //Wait for two checksum bytes
      WaitXSymbols(5); //Wait for two CRC bytes plus some calculation time
      if(ChecksumIsValid AND CRCIsValid){
        if(WaitingForAck){
          if(ReceivedAck) MarkSentMessageAsAcknowledged();
          else MarkSentMessageAsRejected();
        }
        if(ReceivedMessage){
          if(MessageShouldBeAcknowledged){
            SetAckTrigger(Position*ACKINTERVAL + ACK_DELAY);
          }
          UpdateInBufferIndex();
        }
      }else{//The frame was corrupted
        MarkSentMessageAsRejected();
      }
    }else{ //Reception of other frames not currently of any interest }
  }else{ //Reception of frames with other network ID ignored }
}
```

Code 3.18: Acknowledge transmission interrupt [ASRadioNode/node_radio.c].

```
interrupt [SCNT_CMP3] void ackInterrupt(){
  MSC_deactivateCompare3();

  //Copy header byte into framebuffer
  (&TRXFBST)[0] = Ack_Header.phr.byte;

  //Initiate transmission
  GR_setTRX_CMD(TRX_CMD_TX_START);

  //Insert rest of frame
  (&TRXFBST)[1] = Ack_Header.frameHeader.byte;
  (&TRXFBST)[2] = (Ack_Header.node >> 8) & 0x00FF;
  (&TRXFBST)[3] = Ack_Header.node & 0x00FF;
}
```

Code 3.19: Transmission channel interface [ASRadioNode/node_channel_management.h].

```
//------ Transmission cycling -----------------------------------------------
void NCM_sortChannels();

//0 means no more channels are available at this time.
inline void NCM_setNextTransmitChannel(uint8_t msgLength);
```

3.7.7. Channel ranking

To keep track of the channels the node has a specialized channel management module. This module is explained in more detail in section 3.8.5, but the function header used to select a channel for transmission is presented in code 3.19. The function will take the length of the message data as input and set the shared "TransmitChannel" variable to the currently most suited channel. Based on the message length the function determines if there is a chance that the node will not have time to transmit the whole frame before the next beacon. This is critical to avoid both that the node transmits a message that the AP is not able to fully receive, and to prevent the node from transmitting when the AP starts transmitting the beacon, possibly corrupting the beacon. If no channels are currently available given the current position in the beacon sequence, the transmit channel is set to 0. This should be interpreted as a CCA miss, invoking a new random timeout.

By default the function will only consider the three channels with the highest energy readings. These three channels will be returned in sequence upon the first, second and third call of the function. The fourth call will return 0. However, if any of the channels are not available at the given time this channel will be skipped and the next in line is returned instead. Upon successful transmission of a frame the cycling through the available channels are reset to prepare for the next access attempt.

In order for the transmit channel to be set correctly it is important that the channel sorting function is called regularly. Since this function might need to loop through the channels a few times it should be called at a point where timing is not too critical. This is achieved by calling it when an energy measurement is performed on an idle channel. At this point in time the reception of a beacon is not expected, and the energy measurement will only be initiated if no outgoing messages are pending, meaning that the node is to be considered mostly idle.

Code 3.20: Pseudo code for the custom CSMA/CA algorithm [ASRadioNode/node_radio_.c].

```
TransmitProcedure(){
  NCM_setNextTransmitChannel(frameLength);
  while(TransmitChannel){
    SetChannel(TransmitChannel);
    PerformCCA();
    if(CCASuccess) {
      InitiateTransmission();
      return;
    }
    else NCM_setNextTransmitChannel(frameLength);
  }
  SetTransmitTrigger(RandomTimeOut());
}
```

3.7.8. CSMA/CA

Transmission is done according to the CSMA/CA methodology where the nodes will perform a random delay before assessing the channel to see if it is safe to transmit. If this is the case the transmission is initiated, if not, the node steps back for another random period before retrying. In this particular case however there is one major twist to the regular CSMA/CA algorithm as the node will have access to multiple channels. Upon a failed CCA on one channel the node will immediately move to the next in line and retry the assessment. This procedure is repeated until there are no more available channels at the given time. This way the traffic load will distribute evenly among the available channels even when all the nodes are considering the same AP to be their preferred one. Pseudo code for this procedure is presented in code 3.20

3.7.9. Transmission

Once the CCA returns an idle channel the transmission should be started as fast as possible to minimize the gap between the CCA and the signal occurring on the channel. This is necessary to minimize the chance of collisions. With CSMA/CA there is always a certain statistical chance for collisions caused by two nodes assessing the channel in exact parallel and thus starting transmission on top of each other. The larger the gap between the CCA and the transmission, the greater risk there is for this to happen.

Once the transmission is initiated the remaining parts of the frame is inserted into the frame buffer. This can be done in parallel with the transmission so long as any given byte is written before the transceiver attempts to transmit it. Generally this is no problem when transmitting since the microcontroller is a lot faster than the on air bit rate, assuming that there is no time-consuming processing involved.

To insert the majority of the frame after transmission is initiated also saves computation time overall since the alternative would be to insert the full frame before the CCA was completed, thus the work with inserting the frame would be done in vain each time a CCA returned a failure.

3.8. Channel management

Assuming that the core functionality works perfectly as intended the next step is to include analysis of the channels and make the modules automatically avoid noisy or occupied channels. This functionality can be seen to be completely separate from the core functionality as it should have no impact on the functionality in a noise free environment. The main target is to get a system that will detect when a new competing network shows up, and moves away from the afflicted channels without the applications in either end detecting any change in performance.

The AP is the module that decides upon what channel that is to be used. As such this is where the channel quality information has to be located. However the AP is operating on a single channel, with no knowledge of anything that might be happening on the other channels. Even the existence of other AP's is unknown to the AP unless it is explicitly told about it from an outside source. The AP might of course swap to a more or less random channel, and in cases with little to no noise and interfering networks this could work fairly well on average, however the worst case scenario of this approach will be an infinite amount of channel swaps in all situations where more than one channel are unusable.

In order to supply the AP with information there could be a module that scanned all the channels to assess the quality and indicate where other networks are located, both in the form of other AP's and alien networks. In fact this is in part already done by the nodes, they cycle through all the channels sequentially, and all AP's that are within range will be listed within its internal structures in order to be part of the transmission algorithm. By having the nodes perform energy measurements on all channels as they loop through them, they will be able to get a good view of where other interfering networks are located, giving us exactly the information needed by the AP.

Naturally having to relay the information across the radio link is a drawback as it will increase the overhead and directly reduce the overall capacity. Still this is mainly a tradeoff between pure throughput, and robustness and stability. Given that throughput can be increased by adding more AP's, while stability is very hard to improve externally, the choice is simple.

3.8.1. Desired behavior

In a made up scenario an AP is transmitting on channel 7, suddenly a new Wi-Fi network is set up on channel 6. Given that a Wi-Fi channel spans 22MHz while 802.15.4 channels spans 5MHz the Wi-Fi signal will be present, with different strength, in channels 4 to 8. Once the AP detects regular alien transmissions within its frequency band, and potentially an increase in the number of corrupted frames caused by this, it should swap to a new channel that is not polluted by outside noise. If no other networks are within range this can be any channel from 1-3 and 9-16. Assuming there are two other AP's in range, one working on channel 15 and another working on channel 2, these two channels should be blocked by the swapping AP making it choose between channel 1, 3, 9 to 14, and 16.

If two AP's fell victim to the newly arrived Wi-Fi channel, say one AP on channel 7 and one on channel 5, they should both swap. In this case it is desired that the risk of them swapping to the same channel is as small as possible, or impossible if that is feasible. In any case there should be mechanisms that can detect such a collision and correct it without it influencing the system much more than a normal channel swap.

3.8.2. Quality of readings

One thing having a significant influence upon how the AP's sees the channels is the fact that the system will be fairly spread out geographically. This means that sources causing channels to be unusable in one area might not have any influence on AP's covering other areas. Thus an AP is only interested in channel information gathered within the zone where it is the prominent one. This can be achieved by only having the nodes append the channel information when transmitting to their preferred AP, and not on the channels considered to be the second or third in line.

Another important factor is the nature of general radio traffic. Typically it will come in the form of sporadic or regular bursts. This means that a node can pick up noise at one point while at the next sampling the channel appears clean. If only the last sample is to be relayed to the AP it will most likely indicate a clear channel in the vast majority of the cases. This means that the node has to calculate some channel quality factor based on a number of samples giving it some sort of memory on the matter. In this regard it must also be remembered that the nodes will be moving, meaning that the number of samples it evaluates to determine the quality of the channel has to be limited, lest it becomes too inaccurate. The closer together the samples are geographically the better the quality of the measurement will be from the AP's point of view.

3.8.3. Access point view

When the AP receives the readings from the connected nodes it has to choose how to interpret them, specifically it has to decide how to treat multiple, possibly very different, readings of the same channel. One solution is to take the average; however that is not likely to be the optimal solution given the circumstances. The AP needs to look at the worst case scenario for each channel within its covered area. This involves that a node reporting a channel to be virtually unusable while others reports it to be clear should cause the AP to regard the channel to be unusable. This is to prevent spots with high degrees of interference from being lulled down, leading to nodes being trapped inside them, unable to reach any of the surrounding AP's.

To accommodate this, a simple envelope tracking is chosen, this works by applying the new reading to the stored value if it is larger, otherwise the new value is ignored. The stored value, the envelope, is then decreased at regular intervals, meaning that it will gradually decrease if no larger readings are received. How fast the envelope is decreased can be a configurable parameter, however in this case it could very well be decreased quite slowly. This because there might be a fairly long period of time when no nodes move within a given area, meaning that a fast decrementation would cause the readings for that area to drop out of the channel footprint prematurely. Still the decrementation has to be fast enough to cover the possibly fluctuating nature of the radio to provide the desired dynamism.

3.8.4. AP's current channel assessment

The AP also has to measure the perceived quality of its current channel to determine when it's time to swap. This could be done using the ED measurement, but this measurement prevents the transceiver from decoding the signals received for the duration of the test, meaning that it should only be performed when no frame is expected. Since the AP must expect to receive frames at any time during its access window this is not a viable solution. Instead the RSSI register value is used directly, and a number of readings are averaged across each period. This way the AP effectively performs a manual ED measurement. In addition to the energy measurement the AP will listen for corrupted frames, frames with different network ID's, and last but not least, incoming beacon frames.

To estimate the quality of the channel a quality indicator is needed. This estimator needs to be able to indicate then quality of the channel during a relatively short time span, as a sudden drop in quality should be detected as fast as possible. To accommodate this, an inverse envelope is used where a byte is incremented towards its maximum of 255 each time a beacon is transmitted. Upon the detection of an undesired event, be it corrupted frames, noise, or alien beacon frames, the value is decremented by a set, or random, value. If the value falls below a set limit the channel is considered unusable and the AP initiates a channel swap.

Simple noise should generally only cause a minor decrementation. Corrupted frames should cause a not too large, but still significant, decrementation. Frames originating from nodes or AP's with a different network ID should cause a significant decrementation. It is desired to make these decrementations random to prevent two competing AP's on the same channel from swapping at the exact same time. Randomness will make it more likely that one AP reach the limit before the other, making it swap channels and leaving the other alone on the first channel. The remaining AP will then stop decrementing its quality indicator and within not too long the indicator is back at its maximum value.

3.8.5. Node view

The main task of the nodes regarding the channel measurements is to provide the AP with an accurate measurement of the channel footprint at its current physical position. A simple envelope is a possibility here as well, however that might not reflect the channel quality in a good way since single spikes in energy will return the same value as a long sequence of high samples while these two cases will have very different influence on the channel quality.

One possible extension to the envelope, in order to make it give a better idea of the actual channel quality, is to add a filter that will make it ignore high measurements until they come at a rate that is considered problematic. One way to achieve this is by adding a factor that is incremented when a reading is too low to be registered, up to a maximum value. When a high reading is received the factor is decremented by a set amount equal to the spacing at which we allow spikes to come. This way spikes that come further apart than the set spacing will not be able to lower the factor to a level where the reading is actually included in the envelope. When considering if the reading

1.	Single noise sample
2.	Sequence of 16 high samples
3.	Samples increasing from 0 to 20 with 10 high samples.
4.	Five high samples followed by gradual decline to 0.
5.	High readings at regular intervals, more and more frequent until they are continuous. Lasts until all test cases has reached maximum value.

Table 3.2.: Test cases for channel energy registration.

should be applied to the envelope or not, the original reading is divided by the factor plus a constant smoothing value. This will make the envelope gradually rise to the actual value if a long sequence of high readings is measured. The formula can be seen in equation 3.1. The multiplication with the smoothing factor plus one after the division is done in order to make the resulting reading equal to the original once the factor reach its minimum value of 1.

$$newReading = \frac{Reading}{Factor + Smoothing} * (Smoothing + 1) \tag{3.1}$$

The last option is simply to average a set number of readings. This is very straight forward but might require lower limits in order to be used effectively since a channel should be considered noisy even if there are multiple 0 readings.

3.8.6. Node channel assessment model

In figure 3.6 the different solutions are modeled in a spreadsheet, the columns indicated with numbers in parenthesis are envelopes with filtering using the following coding (Spacing, Smoothing, MaxFactor).

The spreadsheet is found in the archive file supplied with the report as "Channel Management Filter Design.xlsx".

During debugging and testing it was seen that noise from Wi-Fi generally caused energy readings in the range of 5-20 when the source was 5-15 meters away from the test modules. Based on this the input data is kept between 0 and 20. The input scenarios are presented in table 3.2. The test cases are placed so far apart that all readings reach zero before a new test is initiated, and the underlying factor of the envelope filter is reset to its maximum value before each test case.

The graph easily shows how the simple envelope triggers on every single noise sample. The average also behaves as expected with a smooth incrementation and decrementation before and after all test samples. For the filtered envelope it is clear that a lower spacing and higher max factor will cause the envelope to stay close to zero even with sporadic noise on the channel. The smoothing factor causes the envelope to gradually rise instead of jumping to a high value once the factor has been decreased sufficiently.

In order to see what approach works best in practice all methods are implemented, the functions can be seen in code 3.21. In order to make them configurable at runtime all functionality for all the modes are included in the compiled version, this should be

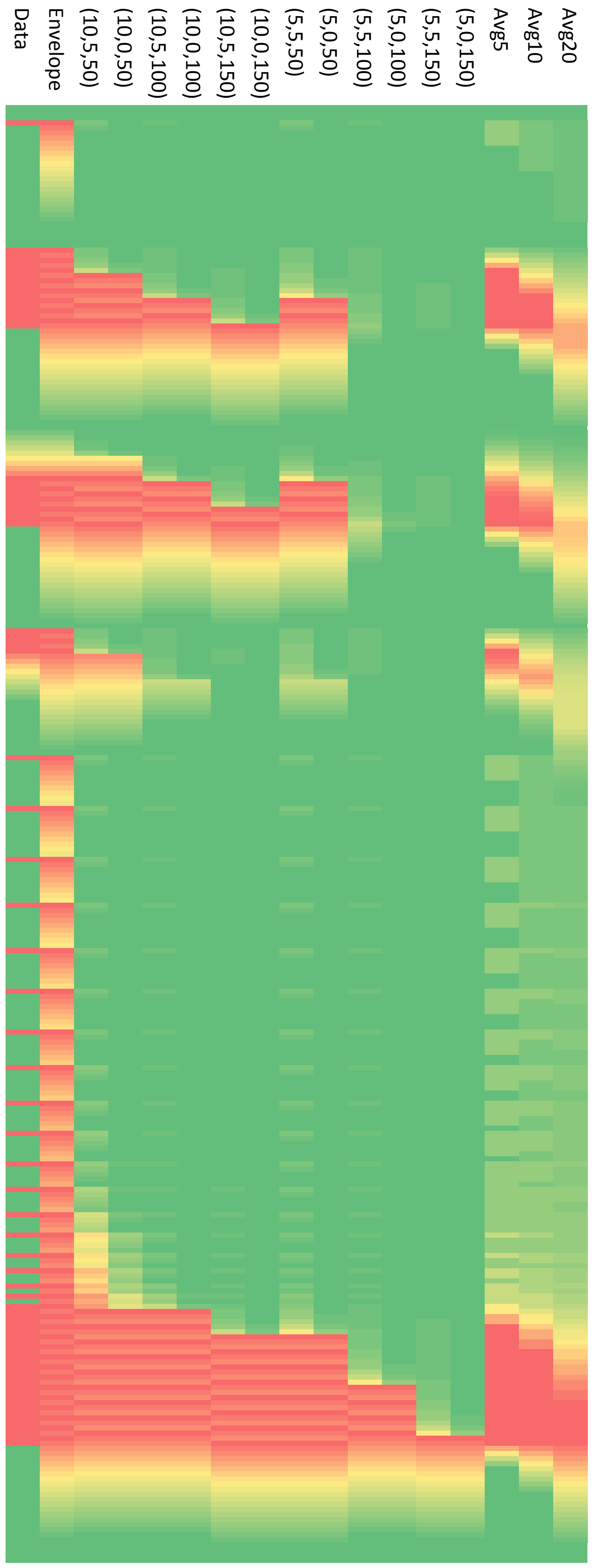


Figure 3.6.: Models of different energy level filters.

changed once a decision has been made on what method to use, to limit the memory usage and program size.

3.8.7. Active channel ranking

In addition to the quality estimation of the unused channels the nodes need to keep track of all the active channels at any given point. Both to prevent the AP from swapping to an occupied channel, and in order to accommodate the custom CSMA/CA algorithm described in section 3.7.8. When reporting to the AP all active channels will simply be encoded with the "CH_USED" status, regardless of their quality and signal strength. Internally however the node will store the signal strength of the channels and rank them accordingly.

The node implements a channel information structure containing a large amount of information. The full structure can be seen in code 3.22, note that a large part of this information is redundant or meant for debugging and statistical purposes and can thus be removed once the protocol behaves as intended. The upper variables are the ones concerning the active channels and are the fields that are actively in use by the protocol during normal operation, in addition to the "ed" field that indicate the energy level of the channel in question. The flags and the different fields are explained in table 3.3.

In addition to the channel statistics structure the node has a separate array with the indexes of the currently best channels. This array indicates the channel that currently has the strongest signal in cell one, the second best in cell two and the third best in cell 3. An index indicates what position in the array that is currently being evaluated and ensures that each call to $NCM\_setNextTransmitChannel(uint8\_tmsgLength)$ returns a new channel until all the enlisted are evaluated. The details of how this incorporates with the CSMA/CA algorithm is described in section 3.7.7.

Code 3.21: ED registration and filtering [ASRadioNode/node_channel_management.c].

```
inline void NCM_clearChannelUpdate(uint8_t ed){
  uint16_t tmpEd;
  if(Channels[BeaconChannel].active){
    if(Channels[BeaconChannel].timer >= ACTIVE_TIMEOUT){
      Channels[BeaconChannel].active = 0;
      Channels[BeaconChannel].ed = 0;
      ActiveChannels--;
    }else{
      Channels[BeaconChannel].unstable = 1;
      CM_ChSet(BeaconChannel, CH_OK);
    }
  }else{
    tmpEd = (ed << 8);
    if(Ch_Mgmt_Mode == 0){  //Default envelope tracking
      if(tmpEd > Channels[BeaconChannel].ed)
        Channels[BeaconChannel].ed = tmpEd;

    }else if(Ch_Mgmt_Mode == 1){   //Envelope filtering
      if(tmpEd > Channels[BeaconChannel].ed)
        Channels[BeaconChannel].edFactor -= EdFacDec;
      if(Channels[BeaconChannel].edFactor > 200)
        Channels[BeaconChannel].edFactor = 1;
      tmpEd = (tmpEd / (Channels[BeaconChannel].edFactor + EdFacAdj)) * (EdFacAdj + 1);

      if(tmpEd > Channels[BeaconChannel].ed)
        Channels[BeaconChannel].ed = ((uint16_t)ed << 8);

    }else if(Ch_Mgmt_Mode == 2){   //Average values
      Channels[BeaconChannel].avgEdSum -=
        Channels[BeaconChannel].avgEd[Channels[BeaconChannel].avgPointer];
      Channels[BeaconChannel].avgEd[Channels[BeaconChannel].avgPointer] = ed;
      Channels[BeaconChannel].avgEdSum += ed;
      Channels[BeaconChannel].avgPointer =
        (Channels[BeaconChannel].avgPointer+1) % AvgNumber;
      tmpEd = ((Channels[BeaconChannel].avgEdSum << 4) / AvgNumber) << 4;

      Channels[BeaconChannel].ed = tmpEd;
    }
    if(Channels[BeaconChannel].ed > (HIGH_LIMIT << 8))
      CM_ChSet(BeaconChannel, CH_BAD);
    else if(Channels[BeaconChannel].ed > (MEDIUM_LIMIT << 8))
      CM_ChSet(BeaconChannel, CH_OK);
    else
      CM_ChSet(BeaconChannel, CH_GOOD);
  }
}
```

Code 3.22: Channel information type [ASRadioNode/node_channel_management.h].

```
//------ Channel data ----------------------------------------------
typedef struct __CH_STATS_EXTENDED{
  unsigned int active   : 1;
  unsigned int unstable   : 1;
  unsigned int inBestList : 1;
  unsigned int netID    : 4;
  unsigned int      : 1;
  uint8_t timer;
//--- CH ranking, configurable ----
  uint16_t ed;          //ED mode 0
  uint8_t edFactor;         //ED mode 1
  uint8_t avgEd[ED_AVG_PERIOD];  //ED mode 2
  uint16_t avgEdSum;
  uint8_t avgPointer;
//--- Statistical data ----
  uint8_t DataPointer;
  uint8_t sent[DATA_GATHERING_PERIOD];
  uint8_t ackMiss[DATA_GATHERING_PERIOD];
  uint8_t csmaMiss[DATA_GATHERING_PERIOD];
  uint8_t crcFailure[DATA_GATHERING_PERIOD];
  uint8_t bcnMiss[DATA_GATHERING_PERIOD];
}__CH_STATS_EXTENDED_t;
```

Field	Data	Use
active	Bit	Indicates that beacons have been received on the channel during the last X periods.
unstable	Bit	Set once an expected beacon does not appear or if a beacon shows up unexpected. Excludes the channel from transmission while still keeping it in active status. This status is cleared once two concecutive beacons has been received on the channel.
inBestList	Bit	Set if the channel is one of the channels currently included in the transmission procedure. The maximum number of channels to include can be configured through defines with the default value being three.
netID	4 bits	The netID of an active channel. Used to distinguish between active channels that might be included in the transmission procedure, and channels that are occupied by other separate networks.
timer	Byte	Incremented each period, set to zero once a beacon is received. Used to determine if a channel should be considered to have gone inactive after a configurable amount of periods without beacons.
ed	2 bytes	The energy level of the channel, reflects the signal strength of the beacon on active channels, and the estimated quality of an idle channel. Scaled up to two bytes to allow better resolution for the idle channel quality estimations.

Table 3.3.: The actively used fields of the channel statistics structure.

3.8.8. Channel info decoding

In order to optimize the transmission of channel information between the nodes and the AP's the information should be as compact as possible. Initially we must have at least three levels, indicating if a channel is in use as an AutoStore channel, if the channel is seen to be completely clear and lastly if the channel appears noisy. Any more states would slide in between idle and noisy to allow the AP to rank the channels better if no truly idle channels are available. Given these requirements the preferred choice is to use two bits per channel, this allows for the complete collection of channel info for 16 channels to fit in four bytes. This also gives one extra state that can indicate medium noise.

Based on the experienced noise levels of 5-20 as stated in section 3.8.6 the limits for channel classification are initially set to an energy level of 3 for medium classification, and 10 for high. In figure 3.7 the modeling done in section 3.8.6 is repeated, but with the resulting values filtered into these three categories, Low noise, medium noise, and high noise.

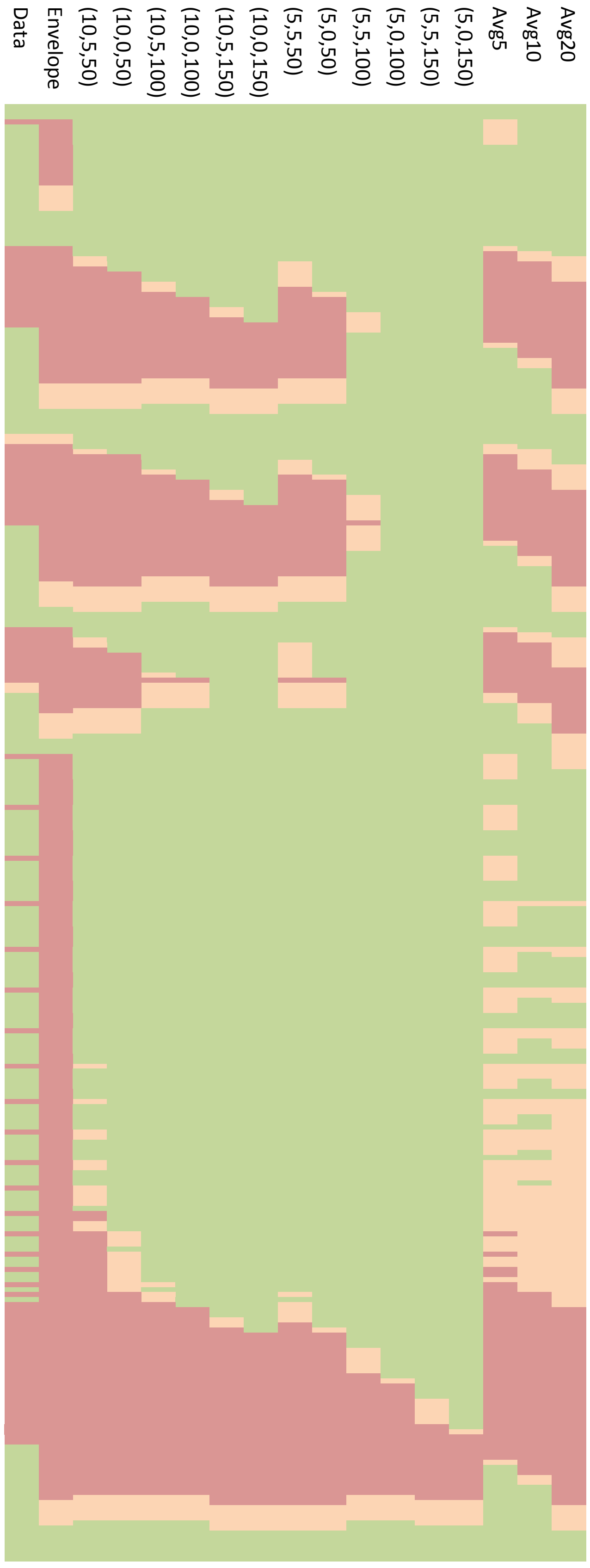


Figure 3.7.: Models of different energy level filters. With values decoded into high, medium, and low.

3.9. Timing and parameter calculations

3.9.1. Core timing

In order for the protocol to work correctly it needs a number of global configurations. At the very core of the system is the major timing constant defining the size of the period, the time between two beacons on a single channel. This number is equal to 62500 [1] divided by the beacon frequency, and rounded down to the closest symbol. However the resulting number also has to be divisible by 16, thus it has to be rounded down to the closest number containing this property.

Next is the subperiod, being the window each channel has for beacon transmission within each beacon period. This number is equal to the period divided by the number of channels, in most parts of the world 16, with some exceptions for local regulations. In practice the protocol will always use 16 channels as the basis for the timing calculations, and in case there are local regulations they will be fulfilled by implementing special blacklists of channels internally in the AP's. Since this is not of any interest for the core functionality however, it is not further elaborated in the work with this thesis.

Thirdly the acknowledge phase has to be split into the desired number of slots. In section 2.5.5 the number of messages per beacon was limited to three, also it was stated that the slots had to be a minimum of 25 symbols long. Given this the slots can attain any value between 25 and one third of the subperiod.

The maximum beacon size must be defined to let the protocol dimension the beacons according to the current timing. This is explained thoroughly in section 2.5.6 and in table 2.9 different values are presented for the different timings. In order to make this parameter easily usable for the protocol it is desired to make it comparable to the index in the frame buffer at which data is being written. This means that the overhead portion covering the checksums, in total four bytes, should be subtracted. In addition to this some effort was put in to minimize the interpretation overhead of the subperiod by having the nodes interpret the beacon as it is being received, as described in section 3.7.4. Still the functionality was not fine tuned and given that optimization can be done at a later stage the maximum beacon size was left at a surely failsafe value.

3.9.2. Beacon frequency

In code 3.23 the major configurations are presented at 31Hz. The reason for choosing this frequency comes from the synchronization of the AP's. Because of the rounding necessary when calculating the period and the subperiod, not all frequencies have their values add up to one second. Generally it can be assumed that having the sync signal arrive up to one half subperiod too late is acceptable, any further and the nodes risk moving on into the next channel, with disconnects as the result. In table 3.4 the period, subperiod and symbol offset from a full second are shown. As seen here 31 is by far the best choice in the rage from 25-35 and have thus been chosen for the time being. By changing the synchronization frequency these calculations can be adjusted accordingly, but given that the synchronization is considered fully external from the protocol 1Hz is used as the basis throughout this thesis.

[1]The number of symbols per second according to the 802.15.4 standard.

Freq	Subperiod	Period	1 sec offset
20	195	3120	100
21	186	2976	4
22	177	2832	196
23	169	2704	308
24	162	2592	292
25	156	2496	100
26	150	2400	100
27	144	2304	292
28	139	2224	228
29	134	2144	324
30	130	2080	100
31	126	2016	4
32	122	1952	36
33	118	1888	196
34	114	1824	484
35	111	1776	340
36	108	1728	292
37	105	1680	340
38	102	1632	484
39	100	1600	100
40	97	1552	420

Table 3.4.: Protocol parameters and synchronization offsets.

Code 3.23: Definitions at 31Hz beacon frequency [ASRadioGeneral/general_radio_config.h].

```
//----- 31 BEACONS PER SECOND -----
#ifdef BEACONS_PER_SEC_31
#define BEACON_FREQ 31
#define PERIOD 2016 //Number of counterticks in a full superframe
#define SUBPERIOD 126 //Symbolcount of beacon or ack period
#define ACKINTERVAL 33 //Size of individual ack window (~1/3 of SUBPERIOD - ACK_DELAY -
     4) (min 22)
#define MAX_BEACON_SIZE 40 //Max size (exluding CRC, FCF and phy-header) max 123 or (
    subperiod/2 - 12).
#endif
```

Code 3.24: Channel constants and evaluation limits [ASRadioGeneral/general_radio_config.h].

```c
//------ General Channel Constants ----------------------------------------
// Channel values
#define CH_FIRST 1 // MUST be > 0 (adjust CH_OFFSET to comply.)
#define CH_LAST 16
#define CH_OFFSET 10  //Added to the channelno before applying to HW

#define TIMING_ERROR_BUFFER 10 // Size of timing error buffer (1=16us)

//Channel statuses
#define CH_GOOD 0
#define CH_OK   1
#define CH_BAD  2
#define CH_USED 3

//Channel evaluation limits
#define HIGH_LIMIT 10
#define MEDIUM_LIMIT 3
```

3.9.3. Channel data and values

In code 3.24 the channel range is specified. When setting the channel through the ATmega128RFA1 registers the values from 11 to 26 has to be used, thus a constant is specified that has to be added to the channel id when applying it in hardware. This also simplifies the task of modifying the protocol to a different hardware at a later stage if that should ever be of any interest.

The parameter "TIMING_ERROR_BUFFER" defines the buffer period handling inaccuracies with the synchronization signal as described in section 2.5.6. With perfect synchronization this value can be set to zero and the maximum beacon size can be increased accordingly.

Lastly the different channel statuses reported by the nodes are specified as well as the limits used when evaluating the channels. As stated in section 3.8.8 the channel report is compressed to two bits per channel, giving four distinct values.

3.9.4. Node specific definitions

The parameters defined in code 3.25 are only used by the node, meaning that any change here will have no influence on the AP so long as it does not cause the node to become unstable. The sync buffer is the offset between the beacon timestamp and the node's beacon timer trigger. This offset is illustrated in figure 2.3 and the value is subtracted from the subperiod, together with the sync error buffer, before the timer is initiated with the resulting value. The value of the offset is the sum of the time needed by the node to swap channels, the reception of the physical header, the header information needed to determine if the frame is indeed a beacon, and triggering the beacon timestamp. In total this equates to a minimum of 3 + 15 symbols with the current implementation.

The acknowledge delay is added to the acknowledgement period to move the first period slightly away from the subperiod trigger. This is to allow the node to complete the full interrupt routine triggered by the subperiod trigger before the acknowledge trigger hits.

The remaining configurations have to do with the channel management. The most

Code 3.25: Node specific definitions [ASRadioGeneral/general_radio_config.h].

```
//--------------------------------------------------------------------------
//    Node constants, any change needs recompilation of Nodes only
//--------------------------------------------------------------------------
#define SYNC_BUFFER 20 //Overhead buffer (min 15+3us = >18)
#define ACK_DELAY 7 //Offset on ack timer relative period trigger
#define TRANSMIT_COUNTER_BUFFER 5 //CCA free buffer at each end of subperiod.
#define IDLE_CHANNEL_CHECK 40 //Offset from ED to period trigger (min 8+processing)
#define TRANSMIT_BACKOFF_FACTOR 8 //backoff = random()%factor - max 16

//---- Node Channel management ----
#define MAX_BEST 3 //Maximum  number of channels to access for transmission
#define ACTIVE_TIMEOUT 5 //Number of silent periods before deactivating channel

#define EDDECAYFACTOR 6 //ED decreased by itself shifted this far right
#define DATA_GATHERING_PERIOD 20 //Number of seconds of history
#define SENT_PER_MISS_LIMIT 33    //Limit for "BAD" channel (1/x = %)
#define SENT_PER_CSMA_MISS_LIMIT 3  //Limit for "BAD" channel (1/x = %)

#define ED_AVG_PERIOD 20
```

important are the "MAX_BEST" that indicate the number of channels the node should attempt to access through its CSMA/CA procedure. The details of this are explained in section 3.7.7.

Secondly the parameter "ACTIVE_TIMEOUT" sets the number of subperiods a channel has to go without beacons before it is considered inactive. Ideally this value should be one, but this would cause the channels to go inactive if a single beacon got corrupted by outside noise. This would again lead to the nodes going into search mode if they were only in range of that one AP. In total this would impact the traffic far more than what one missed beacon would imply.

3.9.5. AP specific definitions

The AP mainly needs configuration of its channel management functionality and synchronization. The RSSI configuration specifies the number of readings to perform and the limit of then to consider the channel to be noisy.

The link-q parameters define the factors used when decrementing the quality indicator. A random value is generated and the quality indicator is decremented by this value modulo the configurable factors.

The configurations are shown in code 3.26. One thing to note is that the channel swap limit is currently set to 0, this was done during testing to disable the channel swapping mechanism as it became clear that there was a major flaw in the conceptual functionality of the quality estimation. The details regarding this problem are discussed in section 8.3.

Code 3.26: AP specific definitions [ASRadioGeneral/general_radio_config.h].

```
//    AP constants, any change needs recompilation of AP's only
//----------------------------------------------------------------------
#define BEACON_BUILD_TIME 40 //AP: Time to prepare beacon frame
#define SYNC_OFFSET_TO_BUILD 20   //Window from sync signal to build trigger
#define SYNC_PERIOD_MAX_OFFSET 3  //Max sync offset for setting timers

#define RSSI_POLLS 16 //Max 32. Should optimally be a factor of PERIOD
#define RSSI_AVG_LIMIT 4 //0,5 * 2^3

#define LINK_Q_COMPETE_FACTOR 64    //Competing frames
#define LINK_Q_FAILED_FRAME_FACTOR 8  //Corrupted frames
#define LINK_Q_SWAP_LIMIT 0//64  //Swap when quality < Limit
```

4. Test preparation

4.1. Main test plan

All major testing should be conducted at Hatteland Computer's offices in Nedre Vats. This way the need for shipping large amounts of equipment back and forth is avoided. The downside is that the testing period is fixed in length and the possibility of extending it in case of problems is limited. However a number of the tests are relatively straight forward once a higher number of radio modules are available, as they are simply expansions of the tests used when debugging. Also the work with the needed framework will be significantly simpler once all the involved parties are at the same location.

The tests that are to be conducted are the following:

- Single AP statistics with different numbers of nodes.
- Single AP statistics during channel swap.
- Multi-AP statistics with different numbers of nodes.
- Multi-AP statistics during channel swap.
- Channel footprint using the different algorithms implemented.
- Effectiveness of automatic channel swapping algorithm.

In order to find the behavior under close to optimal conditions the tests should first be conducted with stationary nodes in a static environment. This will give the most accurate results regarding the throughput and effectiveness of the custom CSMA/CA algorithm. Under these circumstances the results should be directly comparable to the theoretical calculations.

Secondly the same tests should be conducted in a more realistic environment. Hatteland Computer has a medium sized AutoStore installation used for testing and quality control of equipment before shipping it to customers. By mounting "dummy" radios on the robots it will be possible to get realistic movement while not actually using the new protocol to manage the system. This way one can observe how the protocol behaves in the real setting.

4.1.1. Synchronization Signal

In order to conduct tests with more than one AP the synchronization signal has to be present. This involves scaling the current sync signal running at 130Hz down to the required 1Hz. This can either be done at the source, which would be the simplest solution by far, or by having the AP's scale down the signal before relaying it to the protocol. This last approach will require that the AP's have a second sync signal to manage the downscaling. One suggested solution to this is to use a broadcasted TCP/IP message to initiate the procedure. However this will not be 100% failsafe.

One major consideration when choosing what to do is the need for backwards compatibility. This is why downscaling at the source might not be the preferred solution. It is highly desired that the module providing the sync signal is directly compatible to both new and old installations without any need for reprogramming or other configurations.

How this is best done will be discussed and decided during the first few days of the testing period.

4.1.2. AP-PC communication

In the actual installation the communication will go over TCP/IP, with each AP having a unique static IP address. During the work with the protocol the communication was done using a UART interface. This was possible because not all the pins of the radio chip was currently in use, thus soldering on a set of pins and connecting a UART to USB bridge was a simple procedure.

When doing tests with only one AP, UART communication through a terminal will be sufficient as there is no need to choose where to route the information. When doing tests with multiple AP's however a more advanced link is needed to enable dynamic distribution of the messages.

Since TCP/IP is currently used for communication with the AP's there exist software libraries giving quick access to this feature on the PC side. The main work will be on the AP itself since the TCP/IP communication is customized for the old protocol and will most likely have to be rewritten to comply with the new system. This work has to be done eventually if the new protocol is to be incorporated into the system, thus making a firmware prototype for testing should be desired in any case.

With this framework in place it will be possible to make a simple script for sending and receiving messages to and from the AP's and dump the results to a simple text file.

Again the details will be discussed during the first days of the testing period.

4.1.3. Test setup

In order to make the tests easily repeatable a simple test interface is made. This acts as an application on the node and the AP side, sending messages at a rate given by its current test mode and reacting to commands when instructed. With a well made interface it should be possible to program the nodes only once for the entire test period, using commands to change parameters between different tests. The use of the eeprom to store data across hard resets can also be utilized. In particular in the matter of node id's to ensure continuity between tests if some nodes appear to have irregular behavior.

On the AP side the design of the test program depends largely on the presence of a working TCP/IP interface. With this in place the AP will simply relay messages, placing the actual test program on the PC. Without such an interface the AP needs to run the test program itself, sending results and receiving commands via the UART channel. How this is done is of no matter to the results however, as the difference has no influence on the protocol and the traffic "on-air".

The results will be stored to a text file in the form of a comma separated list of values, making the results easy to import into excel or any other number-crunching software.

4.1.4. Testing procedure

With the test program in place the test procedure is simply the act of initiating a preset sequence of commands configuring the nodes to change modes and parameters as the test progress. This makes the tests easily repeatable and easily comparable.

Code 4.1: General portion of test setup definitions [ASRadioGeneral/test_setup.h].

```
#define CMD_RESET 0 //Clear all testdata and testvariables.

#define CMD_FILLERFRAME 1 //Meaningless spam frame..

#define CMD_NODENO_ASSIGNMENT 2
 /*Sent by node to request nodeno, contains four random bytes to assure uniqueness.
 Reply will be: CMD_NODENO_ASSIGNMENT, NODENOH, NODENOL, RANDOM1 - RANDOM4. */

#define CMD_NODE_DATA_1SEC 4
/* Sent by nodes, Seqno + list of values for the previous second.
Spamming same frame for a full second. (Check Seqno to avoid multiples.) */

#define CMD_NODE_CONFIG 5

#define CMD_AP_CONFIG 6 //Not transmitted on air, interpreted by the AP
#define CMD_WRAP_AROUND 7 //Not transmitted on air. Modifies the cmdPtr to enable
    looping.
#define CMD_RESET_AP 8 //Not transmitted on air. Does a simple asm("jmp 0")
```

4.2. Test program

During testing the test program was expanded and adjusted several times as the need for new functionality became apparent and other functionality turned out not to be needed, the following description represents the final version and thus includes all functionality used in the final tests and nothing more.

4.2.1. TCP/IP

As was said in section 4.1.2 the communication channel between the computer and the AP's was an item of uncertainty. The work of creating a functioning TCP/IP channel was discussed and barely initiated, but it quickly became clear that the amount of work needed would be larger than initially estimated. This mainly came from the fact that the old protocol was based on polling, where each frame coming from the server to the AP would be directly replied by the addressed node. The TCP/IP code on the primary AP-MCU reflected this by reusing buffers and frames in a way that made it hard to reuse the code effectively for the new approach that does not involve polling. This, in combination with a high workload within the department, concluded the question and caused the TCP/IP solution to be put on hold for now.

This naturally made the test procedures less simple since the simplicity of running multiple AP's decreased drastically. Still, given an intelligently designed test procedure on each AP the tests can be conducted more or less as planned and the results analyzed.

4.2.2. Test setup definitions and functionality

In code 4.1 the general definitions of the test program are shown with comments. Using these parameters it is possible for the test program to control the nodes by resetting them, silencing them, have them request a new ID and change the channel management parameters as well as the mode of the node. In total this provides an interface with all the functionality needed to run the planned tests.

Code 4.2: Node portion of test program [ASRadioGeneral/test_setup.h].

```
//------------- Node part: --------------------------------
/* Node config parameters */
#define NODE_CONFIG_SET_MODE 1    /* Followed by new mode. */
#define NODE_CONFIG_RESET_ID 2    /* Set mode = NODE_MODE_NOID */
#define NODE_CONFIG_SET_CH_MGMT 3   /* Followed by mode + parameters */
#define NODE_CONFIG_SPAMMER 4     /* Follwed by channel and number of seconds */
#define NODE_CONFIG_SILENCE 5     /* Followed by a number of seconds */
#define NODE_CONFIG_HARD_RESET 255  /* Does a simple asm("jmp 0") */

/* Node modes */
#define NODE_MODE_NOID 0
#define NODE_MODE_RESET 1
#define NODE_MODE_1SEC 2
#define NODE_MODE_1SEC_NOCH 4
#define NODE_MODE_1SEC_CHSTAT 5

/* Logging functions */
inline void TST_ResetData();
inline void TST_Flipp();
inline void TST_Beacon();
inline void TST_RecvDirect();
inline void TST_RecvBroadcast();
inline void TST_RecvOther();
inline void TST_Sent();
inline void TST_SentSuccess();
inline void TST_SentFailure();
inline void TST_Disconnect();

/* Frame generation functions */
void TST_prepare1SecDataFrame(uint8_t *ptr, uint8_t *length);
void TST_prepare1SecDataFrameNoChannel(uint8_t *ptr, uint8_t *length);
void TST_prepare1SecDataFrameChStat(uint8_t *ptr, uint8_t *length);
```

The AP itself can also receive commands either through the test sequence or directly from the UART channel in the form of special command characters.

Node functionality

Code snippet 4.2 defines function headers for the nodes to count the occurrence of certain events, and functions for building the data frames that is to be transmitted. This functionality is composed of data arrays containing two cells. Each call to a counter function will increase the counter for the currently active cell by one. At a one second trigger the function $TST\_Flipp()$ should be called causing the active cell pointer to flip to the other cell and reset the values to zero. The functions building the frames will use the values from the inactive cell meaning that the transmitted data is always the values obtained throughout the previous second. The flip function also increases the sequence counter of the node, enabling the AP to detect when the node has renewed its values. The AP will only print the received values to UART when a new sequence number is received for the given node. In addition to the values obtained through the counting functions some internal values from the protocol can be transmitted. In table 4.1 the data supplied by each function is presented. The variables added following the common values comes from the channel management object of the node.

Function		.P1SecDF	.*NoChannel	.*ChStat	AP
Length		27	11	14	-
Data	0	SeqNo			-
	1	Beacons			Channel
	2	Received Direct			Received
	3	Received Broadcast			PLL offset
	4	Received Other			Active node num
	5	Sent			Sent
	6	Success			Success
	7	Failure			Failure
	8	Disconnects			LinkQuality
	9	Channel mode			TestSeq index
	10	Ch1 ED		Sent2	Ch1 report
	11	Ch2 ED		Failure2	Ch2 report
	12	Ch3 ED		CSMA Miss	Ch3 report
	13-26	Ch4-16 ED			Ch4-16 report

Table 4.1.: Data transmitted with the different frame builder functions. All values except SeqNo are printed over UART when appropriate.

Timestamp	ID	Data[1-X]

Table 4.2.: UART output format.

AP functionality

The AP has equal internal data structures as the nodes, arrays with two cells where only one is active at any given time. Each second the active cell pointer will be flipped to the other cell and the AP will print the collected values to the UART channel with ID 0. In table 4.1 the last column shows the values printed by the AP and the corresponding values printed from the nodes. Table 4.2 show the format of the UART output. Each output is on a separate line making the data easily analyzable at a later stage.

4.2.3. Test command sequence

For managing the tests the AP implements an array structure that allows a simple test programming feature. The layout of this array is seen in table 4.3. Each second a command trigger variable is compared to the current second count, if they are equal the pending command is executed and its delta time is added to the command trigger. Lastly the command pointer is incremented to point to the next command in the queue. This enables the issuing of up to one command per second, which is sufficient for all the planned tests.

In code 4.3 an example test sequence is shown. This sequence will silence all connected nodes for five seconds, two seconds later it will change the mode of the nodes to transmit information with channel statistics before it stays idle until the nodes have transmitted for 250 seconds. Then the AP is reset and the test will start over from scratch.

Index	Meaning
0	Delta time, added to a command trigger to indicate when to execute the next command.
1	Target node if the command is to be transmitted. Can be node ID or 0 for broadcast.
2	Command according to the defines shown in code 4.2.
3-7	Parameters corresponding to the preceding command. Unneeded parameters are ignored on the receiving end.

Table 4.3.: Test command array definitions.

Code 4.3: Example test command sequence.

```
uint8_t cmdList[][8] =
{
  {2, 0, CMD_NODE_CONFIG,NODE_CONFIG_SILENCE, 5, 0, 0, 0},
  {253,0, CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC_CHSTAT, 0, 0, 0},
  {5, 0, CMD_NODE_CONFIG,NODE_CONFIG_SILENCE, 250, 0, 0, 0},
  {5, 255,CMD_RESET_AP, 0, 0, 0, 0, 0},
  {0, 255,0, 0, 0, 0, 0, 0} //TERMINATION LINE
};
```

4.2.4. Node ID assignment

The protocol will always initialize the node ID's to the value 0xFFFF, and assume that the application will set the correct ID before any traffic is initiated. Still the node will be able to communicate as the default ID as well, but there will be a lot of address collisions if multiple nodes are left uninitialized. In order to handle this, a specific procedure was used where the test application on the nodes would transmit ID requests to the AP containing a sequence of random bytes. The AP would reply with an available ID followed by the same random sequence, meaning that even if a large amount of nodes would receive the reply, believing it is addressed to them directly, only the one that recognize the random sequence will actually accept the message and set its ID accordingly.

This method proved very effective and dynamic, and was used throughout the whole process of debugging and preparing the protocol and test procedures.

When the actual tests was to be conducted it was desired to give the nodes static ID's to ensure that the same ID would belong to the same physical module each test run. Thus an assigned ID was stored in nonvolatile memory and the node would only request a new ID if it was instructed to do so or if the memory appeared to have been corrupted.

4.3. Test environment

In order to get the best possible data the tests should first be conducted in a relatively noise free environment. This means minimal Wi-Fi traffic and minimal other communication in the 2,4GHz ISM band. To measure this, a computer with a Wi-Fi analyzer can be used to determine where Wi-Fi channels are located, and the channel measurements of the nodes can be used to indicate if there are other sources, and if they are of any significance. This way it will be possible to see the achievable throughput and the packet loss at different traffic loads and compare this to theoretical estimations. This is mostly interesting for the single AP tests, but also to verify that the throughput is increased as expected when multiple AP's are available for the nodes to access.

Secondly the tests can be conducted in a more realistic environment. This is where tests at the actual AutoStore installation will be interesting. For the tests using a single AP this is not of the outmost importance, but for test with multiple AP's this is essential to see that the traffic from the nodes migrate from one AP to another as the robot moves away from one and closer to the other.

In order to test the channel swap algorithm tests should be conducted both with genuine noise from Wi-Fi, and using a dedicated node to generate noise. Here the main target is to see how much the channel swapping influence the overall throughput. In connection to this the channel footprint of the nodes should also be tested to evaluate the different filtering procedures. To get the best possible results it is necessary to have as equal conditions as possible, making the different footprints directly comparable.

5. Expected results

5.1. Theoretical models

Even if theoretical estimation of system performance is not a part of the original research question it was desired to make some simple models that the measurements can be compared to. The models are not based on any research on this kind of estimators, and are solely based upon the knowledge the writer has regarding statistics. As such it is likely that there exist better, simpler and more extensive models that could have been adjusted to fit this case.

The values that it is desired to model are listed below.

- Average number of packets transmitted by the nodes.
- Average number of packets lost due to collisions.
- Average number of CSMA/CA failures.
- Average number of packets received by the AP.
- Average number of messages transmitted by the AP.

5.2. Parameters

5.2.1. Collision window

In order to calculate the number of collisions a major factor is the collision window, the maximum offset that two accesses can have while still both resulting in a successful assessment.

Based on the chip specifications [3] it is stated that the CCA procedure performs its measurements over $128\mu s$, eight symbols, and that the result is ready after another $12\mu s$ making the whole CCA procedure require a total of $140\mu s$. Since the completion of the CCA is indicated by an interrupt is must be assumed that the interrupt delay also applies, adding another $9\mu s$ to the delay. In addition to this comes the processing delay before the transmission is initiated within the interrupt, this mainly involves checking a few flags and should only require a small amount of cycles, it can be estimated to one half symbol, or $8\mu s$. According to [3] it is then a processing delay of $16\mu s$ before the transmission is actually active "on air". Lastly it must be assumed that the CCA can overlap slightly with a transmission without detecting it given that the result is an average value, this effect will get worse as the two involved nodes are further apart and thus detects each other's signals weaker, this overlap can be estimated to one half

Message size	0-43	44-107	107-Max
Current buffer size	2	3	4
Message size	0-14	15-77	78-Max
Correct buffer size	0	1	2

Table 5.1.: Size of timing buffers to avoid collisions with beacon.

Period length	PL	126
Beacon frequency	BF	31
Period count	PC	16
Blocked access p/period	BAP	10
Node message overhead	NMO	36
CCA length	CCAl	8
AP message overhead	AMO	6
Blocked Periods Transmit	BPT	3
Max CSMA delay	MD	7
Max beacon payload	MBP	37
Node message length	NML	X
AP message length	AML	Y
Access Period buffer	APB	Z
Collision window	CW	3,3
StDeviation	StD	2

Table 5.2.: Numerical parameters for the statistical model.

symbol, or $8\mu s$. In total this equates to a total of $12 + 9 + 8 + 16 + 8 = 53\mu s$, or 3.31 symbols.

5.2.2. Parameter collection

In table 5.2 and table 5.3 the parameters used in the models are presented. The majority of parameters are based on the protocol definitions of the beacon frequency etc. As such they can be considered constant. The only parameters that has to be changed from test to test are the message sizes and the number of periods that are blocked from access by the channel selection algorithm, section 3.7.7. This number depends on the message size of the node, and, as described in section 7.2.3, an error was discovered in these calculations in the code. In table 5.1 both the current and the correct values are presented.

The standard deviation parameter are used in section 5.3.5.

5.3. Estimators

5.3.1. Transactions per second

To find the average number of transactions a simple numerical method is used. First the average delay between two CSMA/CA accesses is found. The upper limit of the average delay does not change as it normally does with CSMA/CA, making this value constant. Secondly the average expected wait time from an arbitrary point in time until the first

Access area per period	AAP	$PL - BA$
Total access area	AA	$(PC - APB - BPT) * AAP$
Total transmission area	TA	$(PC - BPT) * PL$
Total node frame size	NFS	$NML * 2 + NMO$
Total AP message size	AMS	$AML * 2 + AMO$
Average period delay	APD	$MD/2 + (MD/2) * BPA/PC + 0,5$
Average CCA length	ACCL	$APD * PL + CCAl$

Table 5.3.: Parameters calculated using the numerical parameters.

CSMA/CA access is found at different numbers of competing nodes. This value can be found by dividing the average wait time by the number of nodes plus one.

Once this is found it is possible to sum up the average wait times and the time spent transmitting, up till the full available period is used. This way the average number of accesses can be found.

One major flaw with this model is that it will overestimate the number of accesses with low number of nodes. The reason for this is that it does not account for the tail in the probability distribution that falls outside the period. This means that if the model assumes that the wait times will allow all messages to be transmitted, then the resulting estimate is that all nodes will be allowed to transmit. Still the model should get more and more accurate as the number of nodes increases.

In equation 5.1 the total time before X out of N nodes has transmitted is presented. This equation is then used in equation 5.2 to determine the maximum X that gives a result not exceeding the full length of the transmission window. This X is then the estimation of the average number of transmissions given the supplied number of nodes and the specified frame size and access period size.

The solution is not statistically graceful, but it is relatively easy to set up in a spreadsheet.

$$TotalTime(N,X) = \sum_{n=0}^{X-1} \left(\frac{CSMAdelay_{Avg}}{N+1-n} + NFS\right) \tag{5.1}$$

$$\begin{aligned} Transmissions(N) &= MAX(TotalTime(N,X)) \\ &\quad where(TotalTime(N,X) < TA) \end{aligned} \tag{5.2}$$

5.3.2. CSMA/CA misses

With a properly made CSMA/CA algorithm it can be assumed that the accesses are truly random in time, meaning that the probability is uniformly distributed across the whole period. Since the result of the estimation is simply whether the channel is busy or not this means that the percentage of accesses resulting in a failure will directly correspond to the current channel utilization.

With the size of the transmitted frames being known, as well as the total size of the period it is possible to use the transaction estimator to estimate the channel utilization and thus, indirectly, the number of missed CCA's.

$$P_{CSMAMiss}(N) = \frac{Transmissions(N) * NFS}{AA} \tag{5.3}$$

5.3.3. Collisions

A collision will occur if two or more nodes triggers their CCA within only a few symbols of each other. This will cause them to gain access to the channel in parallel if it is initially idle, with the result that they transmit on top of each other. Depending on the signal strength, and signal quality, the receiver might be able to decode one message, or all the colliding transmissions can be lost.

In order to estimate the number of collisions we first find the total number of expected accesses per period, this equates to the number of nodes plus the number of extra accesses caused by CSMA misses, the total number is given in equation 5.4.

Secondly the number of possible choices has to be estimated, this is where the collision window comes to play. Since an access will result in a collision if it is made within "collision window" symbols of another access we should divide the total number of accessible symbols by this number. Still this is not exactly correct since the window expands in both directions. In addition the collision windows of two accesses can overlap without problem so long as the access itself is not within the window of the other. As a simple approximation to this the total number of accessible symbols is divided by one and a half collision window. This is shown in equation 5.5.

To find the expected number of collisions with N nodes we use the following reasoning; The probability that one access does not collide with a specific other are $(1 - 1/AccessSlots)$. The probability that neither of the N-1 other accesses collide with the specific one is $(1 - 1/AccessSlots)^{N-1}$. This means that out of the N performed accesses it is expected that $N * (1 - 1/AccessSlots)^{N-1}$ accesses are non colliding, meaning that $N - (N * (1 - 1/AccessSlots)^{N-1})$ accesses are expected to collide with another access. This formula is represented in a clean form in equation 5.6.

$$AccessNum(N) = N + N * P_{CSMAMiss}(N) \tag{5.4}$$

$$AccessSlots = \frac{AccessWindow}{CW * 1,5} \tag{5.5}$$

$$Collisions(N) = N * (1 - (1 - \frac{1}{AccessSlots})^{N-1}) \tag{5.6}$$

5.3.4. AP reception

The number of frames received by the AP is simply the number of transmissions made by the nodes, minus the number of messages lost due to collisions.

$$AvgReceived_{AP}(N) = Transmissions(N) - Collisions(N) \tag{5.7}$$

5.3.5. AP transmission

In order to find the average number of transmitted messages from the AP things gets more complicated. This value will depend on the number of acknowledges that has to be inserted, a number that is equal to the number of messages the AP received during the previous period. The AP can then fit anywhere from zero to three messages in the AP depending on the amount of remaining space. Thus the question is really a question of the probability distribution of the number of incoming frames among the estimated average.

To begin with the remaining beacon space on average with a given amount of nodes are calculated, this involves subtracting the space needed by the incoming acknowledgements from the total amount of available space. This is seen in equation 5.8.

Two approaches are attempted. The first one simply involves dividing the remaining beacon space, after adding the average number of acknowledges, by the space needed per message. This should give a reasonable result when the number of nodes gets larger. However this does not take into account that the realistic probability distribution of the number of messages would be some sort of exponential distribution. This means that it will most likely overestimate the number in many cases, in particular when the number of nodes are low.

The second option is to assume that the number of messages received is normally distributed around the average number. By assuming a standard deviation it can then be estimated how likely the AP is to fit 1, 2 or 3 messages in the frame. This could provide a reasonable model, but it might be very hard to find a reasonable value for the standard deviation, making the model less reliable. When representing this estimator the expression $\mathcal{N}(x, \mu, \sigma^2)$ is used to represent the normal distribution.

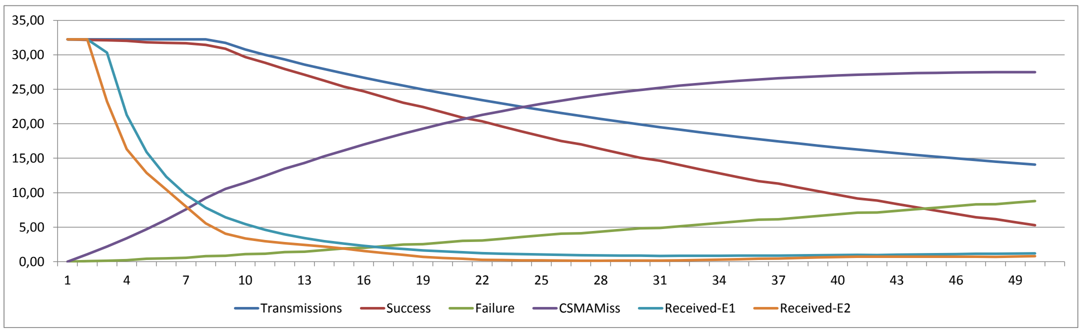
Since both these models are based on the estimated number of messages received by the AP each period, given that they depend on the numbers of acknowledgements, the expected overestimation of the node transmissions will be expected to cause an underestimation with both these models. Still the results should give a reasonable indication overall.

$$RBS(N) = MaxBeaconPayload - AvgReceived_{AP}(N) * 2 \tag{5.8}$$

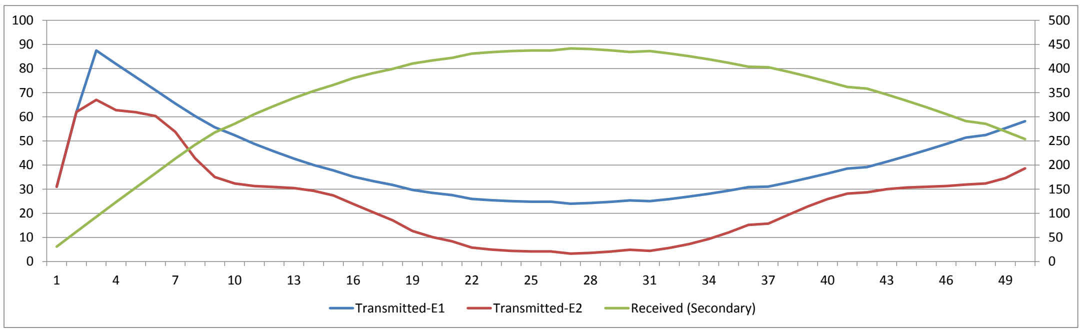
$$MessageE1(N) = MAX(\frac{RBS(N)}{AMS}, 3) \tag{5.9}$$

$$MessageE2(N) = \sum_{m=1}^{3}(1 - \int_{0}^{m*AMS} \mathcal{N}(x, RBS(N), StD)\mathrm{d}x) \tag{5.10}$$

After some simple trial and error the standard deviation was set to two. The same value are used for all three message counts even if that is most likely not fully realistic. It is most reasonable to believe that the standard deviation gets bigger as the number of messages increase. Another factor that is not included is the fact that there is a maximum number of incoming messages equal to the number of nodes. This means that the probability distribution will be cut off at that number, resulting in a corresponding increase in probability over the rest of the available possibilities.



(a) Node



(b) AP

Figure 5.1.: Modeled behavior at different number of nodes using 8 byte messages in both directions. Numeric results can be seen in A.1.

All in all, the estimator is not believed to be particularly good given the high amount of inaccuracies and possible error sources.

5.4. Calculations

As a test case the message sizes from the protocol requirements in section 1.3 are used and the protocol configuration kept as shown in table 5.2. This means a message size of eight bytes for both nodes and AP's. In table 5.1 this is seen to give zero extra buffers if the calculations are done correctly. With these parameters entered into the spreadsheet the result is as seen in figure 5.1.

As expected the total throughput is seen to reach a maximum value before declining as the number of nodes increase further. This is due to the increasing number of packets lost through collisions as the channel gets more and more frequently accesses. This increase is further empowered by the fact that more nodes will result in a better utilization of the channel, causing more failed CCA's which in turn further increase the total number of accesses.

According to these calculations it should be possible to reach a throughput of 440 messages per second from the nodes to the AP, with the requirements in section 1.3 stating that one node will want to send a maximum of seven messages per second this will mean that a single channel has theoretical capacity to handle around 62 nodes at peak traffic. How this holds in practice remains to be seen.

In appendix A the numerical results of the calculations are given. The spreadsheet used when calculating is supplied in the archive file as "Theoretical performance model.xlsx". The spreadsheet performs the calculations, a conversion to one second measures, and a conversion to results that are easy to insert into the spreadsheets containing the actual test results.

6. Test phase

As one could, and should, expect, the testing did not go quite according to plan. Still the majority of tests were conducted in some way, and there are strong indications as to how the last tests would behave had they been performed according to plan. Also the problems that came to view are logical and should be correctable given certain changes and additions to the protocol functionality.

6.1. Problems and delays

6.1.1. Discarding TCP/IP

As explained in section 4.1.2 it was decided to discard the TCP/IP link and simply use the UART communication that had been used during debugging. The reason behind the decision was that it would require more time to get the TCP/IP framework up and running, then the time needed to adjust and reprogram the test procedures to use the UART. Still the overall extra work needed was higher than expected beforehand, causing the testing to be delayed slightly.

6.1.2. Synchronization signal

The synchronization signal had to be adjusted from 130Hz to 1Hz. As stated in section 4.1.1 the optimal way to do this was not clear, but since the actual approach is not of any importance to the protocol it was decided to simply adjust the frequency at the source temporarily to allow the tests to be conducted.

Unfortunately the person responsible for the module supplying the signal was away on vacation during most of the testing period, meaning that the author had to dig into the firmware to modify the signal himself. This made the process a bit more time consuming than expected, causing the testing to be delayed further. As the testing commenced it also became clear that the synchronization signal had some flaws, assumingly in connection to the wraparound of the timer. This caused the one second period to suddenly increase to around one and a half second for a single trigger, leading to erroneously high data readings from the AP. Still this event is easily identified in the results and as such is not considered critical.

6.1.3. Synchronization function

Once the synchronization signal was in place another problem became clear. The synchronization function of the protocol was in no way working as intended. During debugging all tests had been run with only a single AP, and the synchronization function had been tested by triggering it according to a 1 second hardware timer within the microcontroller itself. Once the function was being triggered by the distributed signal it appeared to behave quite different than expected, the errors had simply not been visible with the setup used while debugging.

As a result of this the synchronization function had to be reworked. Instead of the original static offset an implementation as a simple software PLL was attempted. Since time was of the essence it was not possible to create a very advanced function and the last version of the synchronization caused the AP to drift an estimated $\pm 6 - 7$ symbols away from the correct timing.

During the work with the synchronization signal and function the code was debugged and tested using two AP's and 20 nodes. All nodes were continuously in full transmission mode causing them to attempt to transmit each period. The observed behavior with the inaccurate synchronization was exactly as expected. For a period both AP's would receive a significant amount of traffic, at a more or less random point in time one AP would suddenly stop receiving messages altogether while the other clearly received more than usual. This situation would continue for a short period until the traffic suddenly spread across both AP's once again. This process then repeated itself over and over, usually by having the AP's fall into isolation every other time.

In section 2.5.6 the reason for this behavior is described, and the addition of the timing buffer is explained. Still it was obvious that the 10 symbols added as a buffer was not enough with the overall synchronization system being as inaccurate as it turned out to be. Given the assumed offset of $\pm 6 - 7$ it can be assumed that the system could have been stabilized by increasing the buffer to more than 14 symbols, still this was not done because of the event described in the following section. The synchronization issues are further discussed in section 8.2.

6.1.4. Equipment breakdown

As if the added workload was not enough another catastrophe occurred during the work with the synchronization. Following a short circuit all UART bridges got parts of their circuitry burnt. The resulting situation was that out of the original three bridges, one had to be discarded, one only had its TX channel operational, and one only had its RX channel operational. In essence this meant that it was only possible to communicate with one module at the time, making testing with multiple AP's effectively impossible. A set of new bridges was immediately ordered, but they did not arrive until after the test period was complete and the author had left the test location.

6.1.5. Testing at an actual AutoStore installation

With the problems described in the above sections the test period shrunk significantly, and several tests became impossible to execute effectively. Preparations for testing at the actual installation was initiated, but at some point it became clear that it would be better to perfect the tests that was executable at the time, rather than spending a lot of time and effort on preparing tests that was known not to work as originally intended. As such the testing on a physical installation was put on hold for the remaining duration of the thesis work.

6.1.6. Channel swap oversensitivity

The very first tests that were performed used a single AP and, assumingly, a single channel. These tests were mainly meant to see that the core functionality of the protocol worked also after adding a much larger number of nodes than what was used during debugging. At first the tests appeared to work as intended. However once the results were analyzed more thoroughly it was discovered that the AP was not stationary on one channel as intended, rather it was swapping channels increasingly often as the number of nodes increased. Since there was no registered outside noise this indicated that the channel swap algorithm was triggered by events that should not really cause it to trigger.

The reason for the problem quickly became clear and revealed a fundamental flaw in the functionality. The reason for the swapping was the increased number of corrupted frames caused by legitimate CSMA/CA collisions as the number of nodes got larger. Since the AP assumed that all corrupted frames was caused by outside noise this lead to it assessing the channel as increasingly noisy as the number of nodes, and thus the number of collisions, increased. This artifact essentially made the channel swap algorithm unusable in its current state.

6.1.7. Node deadlock

During the late debug and testing stages one problem with the nodes became apparent. After a random time they tended to go into a deadlock making them unresponsive. The timing could be anywhere from a couple of hours, up to several days. At one point 20 nodes were left running over a weekend, after nearly four days one single node was left operational, all others had deadlocked at some point.

Since the error generally took hours to happen it had no impact on the shorter tests, and even on the tests lasting up to one hour, like the channel footprint tests. Because of this the effort put into the issue was also limited, however towards the end of the test period some spare time was found while waiting for the longer tests to complete and the issue was investigated.

Through debugging with JTAG the error was narrowed down to the symbol counter, more accurately one of the comparators stopped triggering even though all registers and the system state indicated that it should, even manual manipulation of the registers through the JTAG interface did not change the issue. The trigger in question was the frame transmission trigger, meaning that the node got into a state where it never actually got to perform the CCA and transmit a frame, even if it were receiving beacons normally and even acknowledging any incoming messages.

In an attempt to narrow further in on the reason for the error, two of the symbol counter interrupts were swapped. This change should theoretically have no impact whatsoever on the functionality, but it did in fact appear to make the nodes more stabile. The change was made at a late stage in the test phase, meaning that there was limited time to do thorough tests afterwards, but during the time that was left no nodes was observed to deadlock.

The reason is still unknown, and the most likely explanation is that there are errors in other, completely unrelated, parts of the code. This can either be pointers that are not set correctly, or buffers that overflows, causing the program to corrupt unrelated data. Because of this it would be wise to investigate the issue further at a later stage, and maybe also do a complete review of the whole code to verify that all buffers and pointers are used correctly.

7. Results

7.1. Test setup

In figure 7.1 an image of the test setup used throughout all the following tests is shown. In the high center of the image is the module supplying the AP's with power. This module is also the one supplying the synchronization signal as all AP's are connected directly to this module for the emergency stop functionality. In the lower parts of the image are all but one node, and to the left two AP's can be seen as well as the last node connected to the computer with JTAG for debugging.

One thing to have in mind is that the nodes are located very close together. This was necessary to get a practical solution with the power supply, but naturally it will cause them to receive each other's signals very strongly. This must be seen as a possible error source, and is seen to influence the results in the channel footprint tests in section 7.9.

In total 21 nodes and two AP's were available for testing. After the loss of UART bridges one AP was reprogrammed to be used as a node, giving a total of 22 nodes and one AP.

During the whole duration of the test phase there were observed to be three Wi-Fi AP's supporting the same network on channel 1. In addition to this another network occacionally showed up in channel 3, however this was so weak that it is unlikely to have any influence. A screenshot of a network scan can be seen in figure 7.8.

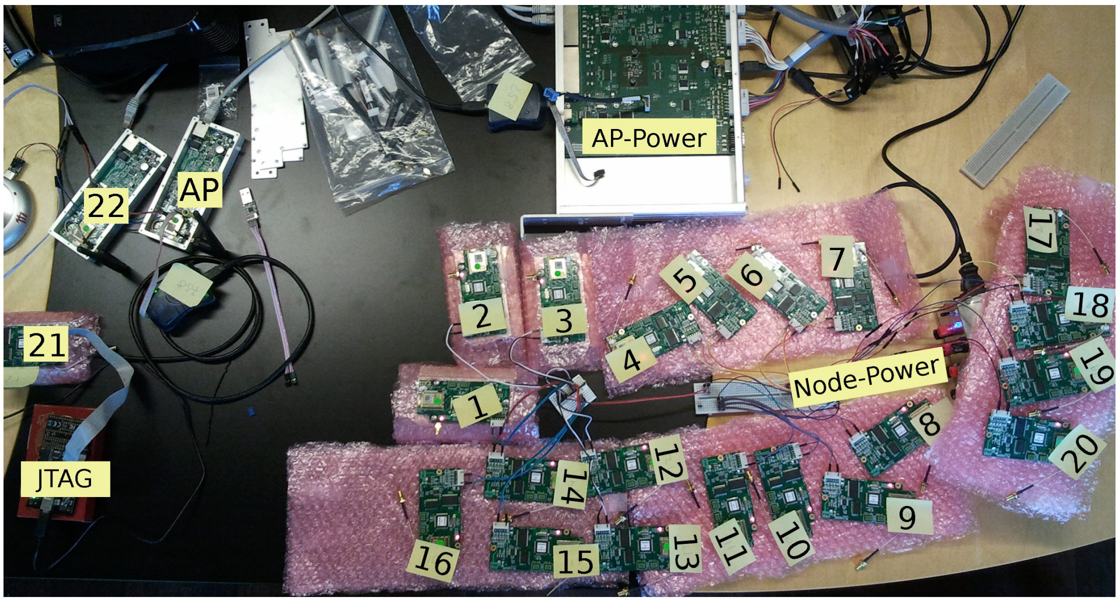


Figure 7.1.: Test setup

7.2. Test result adjustments

7.2.1. Node measurements

During the writing of this report and the connected analysis of the test results one error was detected that had been overlooked during testing. The timing of the test application on the nodes was not correct.

The test application on both the nodes and the AP's are controlled by a hardware timer supplying the application with a signal each second. This then cause the data gathering functionality to refresh the data to be transmitted, and restart the data gathering for the next second.

While the counter on the AP triggers correctly, or at least with such a small error that it is not effectively detectable, the counter on the nodes turned out to trigger an estimated 40ms to late each period. This cause the nodes to report data gathered throughout 1,04 seconds as if the data was actually gathered through exactly one second.

This behavior can be seen in two different ways in the test results. Firstly each node will miss in the list of reports every 25 seconds since its 1.04 seconds then encapsulates the true one second of the AP. This has no impact on the calculated values though since the average values over all reporting nodes are used. Secondly the number of received beacons reported by the nodes will average to 32.24 instead of the correct 31.

Since the error is equal for all data reported by all nodes, the reported numbers can be compared directly, as such the graphs presented are of the correct shape, but all values reported by the nodes should be reduced by 4% to get the numerically correct value. Since values from the nodes and AP's are plotted in separate graphs this only has to be taken into account when comparing the values in the two graphs. For instance the number of messages successfully transmitted by the nodes multiplied with the number of nodes will give a sum that is 4% higher than the corresponding number of received messages reported by the AP.

In order to match the theoretical values and the measurements the theoretical values are adjusted to correspond to the actual timing used by the nodes before they are plotted in the same graph.

7.2.2. Channel statistics

In addition to the erroneous timing, some concurrency issues occurred when reporting the internal values of the protocol regarding channel statistics; the number of accesses, the number of failed frames and the number of CSMA/CA misses. Due to the fact that these numbers are modified within the interrupts of the protocol, while they are read through a non atomic function in the test application there are some cases where the reports are all zeros. This happens when the protocol gets to modify the index and reset the values between when the application reads the index and when it reads the values.

To avoid these readings polluting the results by giving an unrealistically low average they are simply masked out in the spreadsheet, removing the readings altogether. This is simply done by replacing the value in the cell with a blank string if all three values are zero.

7.2.3. Beacon collision avoidance

As described in section 3.7.7 the nodes will be prevented from transmitting on a channel if there is a chance that they will be unable to complete the transmission before the AP deactivates its transceiver in preparation for the beacon transmission. The functionality calculating the blocked subperiods based on message length was later discovered to be quite wrong. Parameters representing bytes and symbols had been mixed, while the resulting sum had been used as if it represented a number of bytes, this lead to the calculated number being significantly higher than intended, causing the nodes to be blocked from the channel too early. After reviewing the existing code and the correct version it was found that for the tests where the nodes transmit 14 or 11 bytes of data two extra subperiod will be blocked. In the tests where 26 bytes are transmitted, one extra period is blocked. Mainly this means that the throughput is lower than it should have been in the afflicted tests, and also that the theoretical estimations must be configured accordingly to correspond to the actual behavior of the physical system.

7.2.4. Test result files

Given the size of the raw test results they are pointless to include as appendices. Still all the presented test results are found in the supplied archive file with names corresponding to the names used throughout this chapter. Also the corresponding excel spreadsheets performing the calculations and graph generation are presented, these as well with the corresponding names.

The tests presented throughout this chapter is only a collection of the total amount of conducted tests, but the ones presented are either a random test among many equal, or the test that best illustrates the conclusions that can be drawn from the given tests. In the archive file all successful tests are supplied for reference. Also note that not all other tests have a dedicated spreadsheet, as the similar results was usually imported into the same spreadsheet when analyzing them.

7.3. Synchronization test

Description In figure 7.2 a test run using the last version of the new synchronization algorithm is shown. The test is conducted using the test program shown in code 7.1 and the parameters shown in table 7.1.

Synchronization issues The major points to note are the increased readings from the AP at time 6 and 142. These come from the known artifact of the synchronization signal and should thus be ignored. Secondly it is a significant amount of disconnects caused by the inability of the synchronization function to keep the timing stable. The value "SyncOffset" in figure 7.2(a) shows the dynamic factor calculated by the synchronization function. This value is seen to oscillate around 3-4, with disconnects mainly occurring when the oscillation reach a top or bottom.

For the AP disconnects are seen as a sudden decrease in received messages and a decrease in successfully transmitted message with a corresponding increase in transmission failures.

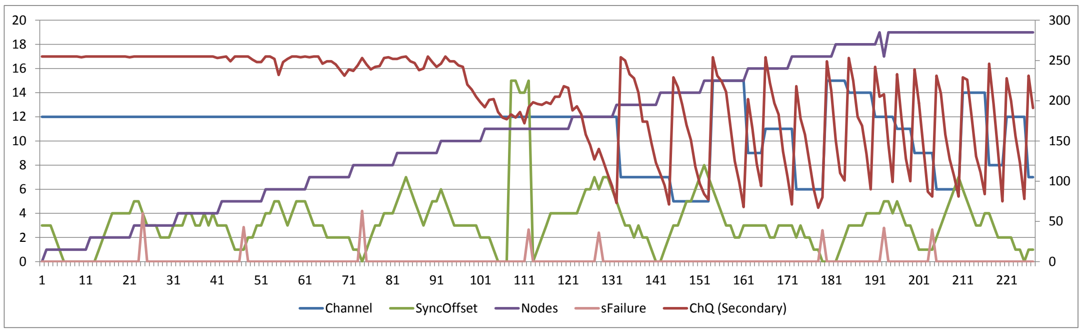
Channel swap issue Another problem clearly visible in this test is the errors with the AP's channel assessment algorithm. The value "ChQ" in figure 7.2(a) show the estimated quality of the channel as a value ranging from 0 to 255. If the value drops below 64 the channel is considered unusable and a channel swap is initiated. The graph clearly shows how this value is very smooth to begin with, indicating a perfectly good channel. As the number of nodes increases the value gets rougher and rougher before it starts to decline consistently. Once the value is low enough the AP swaps channel and resets the value, however the whole process simple repeats itself faster and faster as more nodes are added.

Code 7.1: Test program: Testresults20Nodes10Sec3-16-05-13

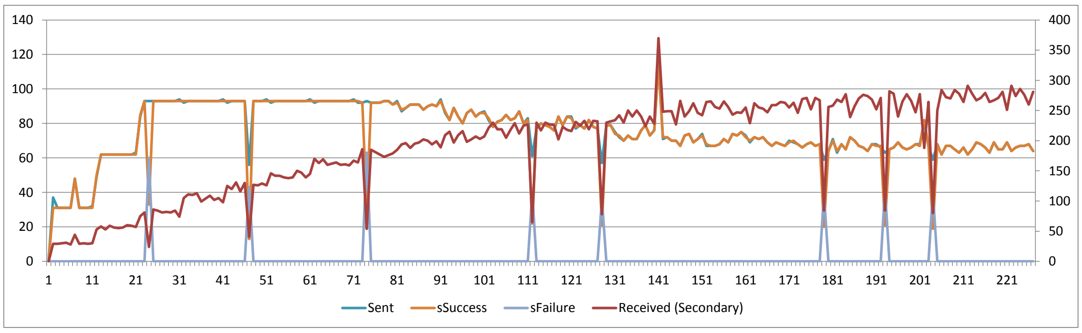
```
uint8_t cmdList[][8] =
              {{7,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},
              {3,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC,0,0,0},
              {10,1,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
              {10,2,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            //-- 3 TO 18 EDITED OUT FOR READABILITY --//
              {10,19,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
              {10,20,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
              //MUST BE TERMINATED BY THE FOLLOWING ARRAY!
              {0,255,0,0,0,0,0,0}};
```

Node message size	AP message size	Node count
26 byte	5 byte	1-19 (20'th node unresponsive)
Test length	Channel Swapping	Synchronization
230 seconds	Active	Active

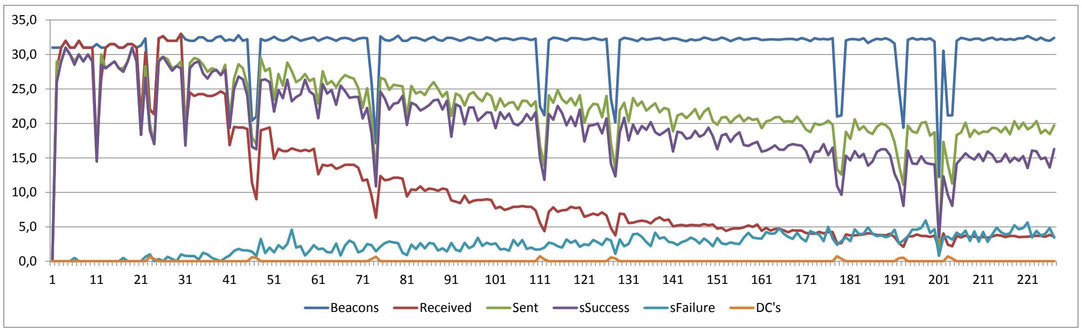
Table 7.1.: Major test parameters: Testresults20Nodes10Sec3-16-05-13



(a) APCH



(b) AP



(c) NODE

Figure 7.2.: Testresults20Nodes10Sec3-16-05-13

7.4. Synchronization disabled

Description Following the combination of more or less catastrophic problems, with the last and most critical being the destruction of the UART bridges, synchronization was simply disabled for the remaining tests. For all tests using a single AP there is no need for synchronization anyways and had it worked as intended it would not have influenced the results whatsoever. Thus the removal of the whole functionality can be considered a reasonable simplification under the current circumstances. In figure 7.3 the test from section 7.3 is rerun with the synchronization disabled.

AP transmissions One interesting, and expected, phenomenon that can be observed is how the number of transmissions made by the AP will have a shape that is to some extent the inverse of the number of received messages. A very good example of this is located at time 120 to 130, with the number of received messages dropping slightly, giving a notable increase in the number of transmitted messages. This is exactly as expected given the way the AP inserts its messages, less incoming frames leads to fewer acknowledgements giving more room for data.

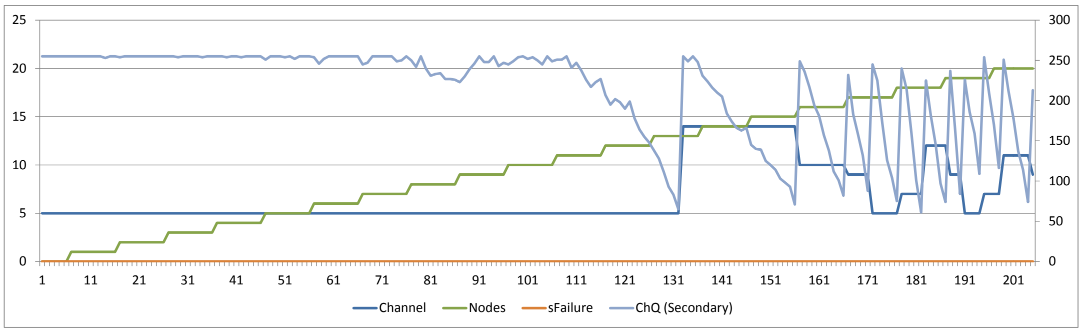
Channel swap issue With the synchronization out of the picture the results gets significantly smoother and closer to the expected results. Still the channel assessment is not behaving as intended and is causing a large number of unneeded channel swaps when the number of nodes increases beyond 11-12.

Code 7.2: Test program: Testresults20Nodes10Sec4NoSync-16-05-13

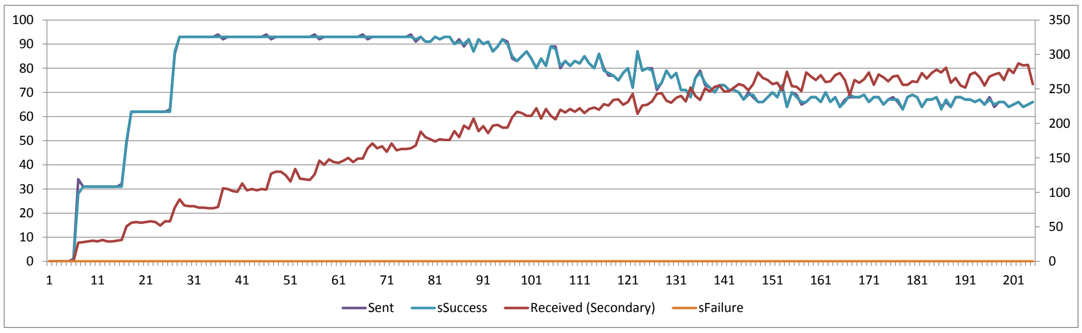
```
uint8_t cmdList[][8] =
              {{7,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},
              {3,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC,0,0,0},
              {10,1,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
              {10,2,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            //-- 3 TO 18 EDITED OUT FOR READABILITY --//
              {10,19,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
              {10,20,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
              //MUST BE TERMINATED BY THE FOLLOWING ARRAY!
              {0,255,0,0,0,0,0,0}};
```

Node message size	AP message size	Node count
26 byte	5 byte	1-20
Test length	Channel Swapping	Synchronization
210 seconds	Active	Disabled

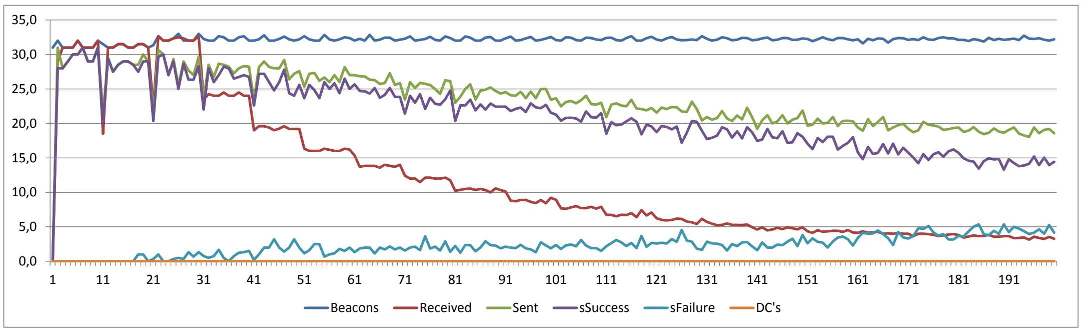
Table 7.2.: Major test parameters: Testresults20Nodes10Sec4NoSync-16-05-13



(a) APCH



(b) AP



(c) NODE

Figure 7.3.: Testresults20Nodes10Sec4NoSync-16-05-13

7.5. Synchronization and channel swap disabled

Description With the channel assessment algorithm also malfunctioning, the automatic swapping of channels was simply disabled. This was done by setting the limit at which the channel swap should trigger to zero, meaning that it would never happen. In addition to this the nodes were reconfigured to stop adding the channel information to their messages, this in order to make the message size smaller and closer to the 8 bytes stated in the requirements. Lastly the messages transmitted by the AP were adjusted up in size from 5 bytes to 8 bytes to comply with the requirements.

With the system in a stabile and good working condition it is also possible to compare the results to the theoretical calculations. To do this the model was adjusted to use the same parameters as was used in the test, and the calculations were formatted in a way that made it possible to plot them side by side with the measured results.

The nodes do not report their number of CSMA/CA misses with the current configuration. Still the estimated number is plotted for completeness.

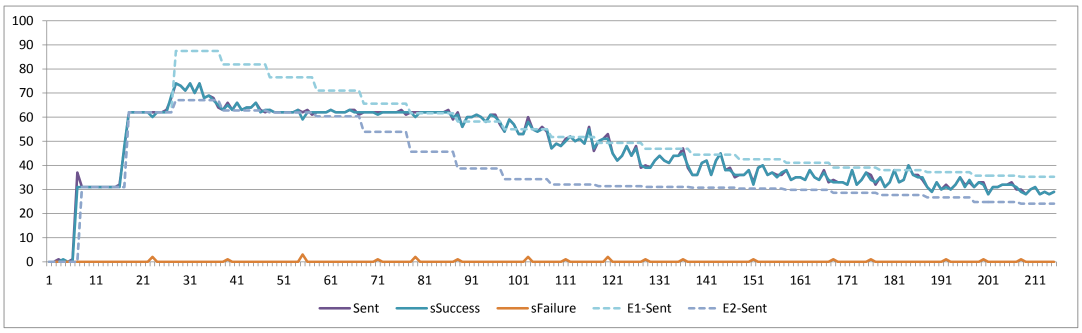
Minor artifacts In figure 7.4(a) it can be seen that there are some occasional failed frames, the source of these are unknown, and with the current test application no more information about the matter is obtainable. Therefore it is not known if the error is caused by nodes not receiving the beacon, nodes not transmitting acknowledge correctly, or the AP not receiving the acknowledge properly. This issue is further discussed in section 8.5.

Code 7.3: Test program: Testresults21Nodes10sec1NoSyncAbsNoChSwap-21-05-13

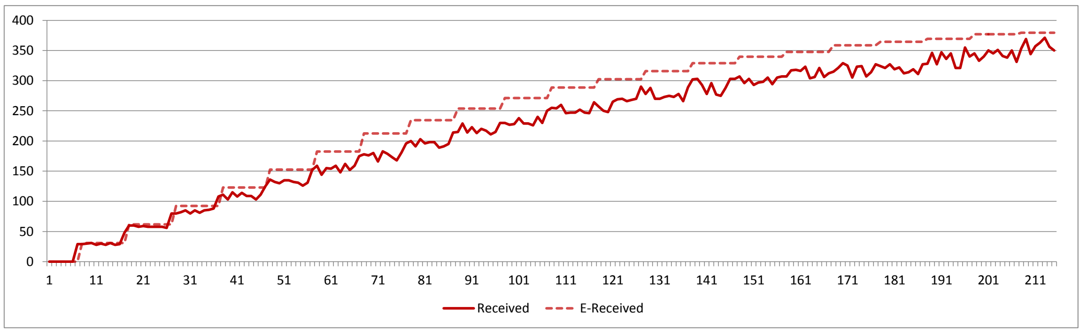
```
uint8_t cmdList[][8] =
            {{7,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},
            {3,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC_NOCH,0,0,0},
            {10,1,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            {10,2,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
        //-- 3 to 19 EDITED OUT FOR READABILITY --//
            {10,20,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            {10,21,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            //MUST BE TERMINATED BY THE FOLLOWING ARRAY!
            {0,255,0,0,0,0,0,0}};
```

Node message size	AP message size	Node count
11 byte	8 byte	1-21
Test length	Channel Swapping	Synchronization
210 seconds	Disabled	Disabled

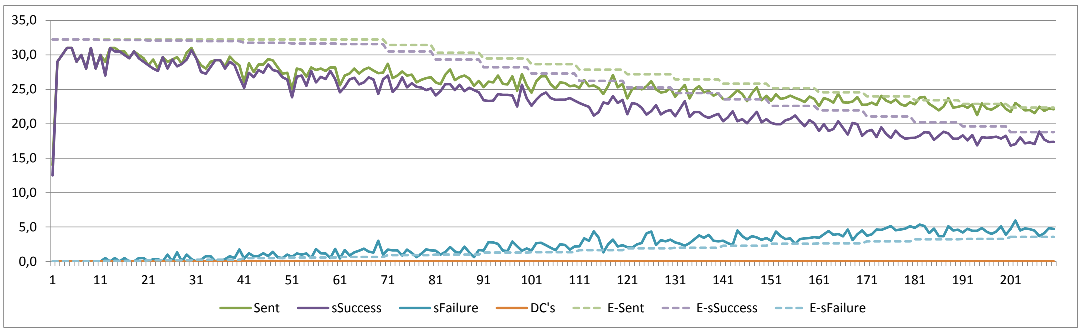
Table 7.3.: Major test parameters: Testresults21Nodes10sec1NoSyncAbsNoChSwap-21-05-13



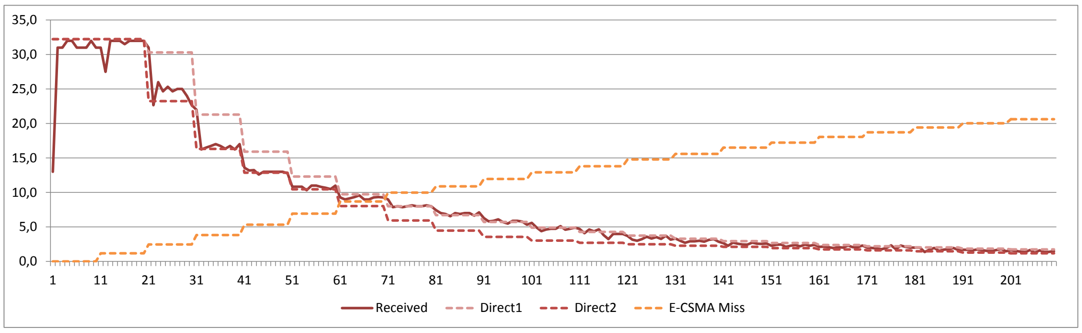
(a) AP Sending



(b) AP Receiving



(c) Node Sending



(d) Node Receiving

Figure 7.4.: Testresults21Nodes10sec1NoSyncAbsNoChSwap-21-05-13

7.6. Multi-AP test (model)

Description As the UART bridges needed to perform the testing got destroyed during the debugging of the synchronization, described in section 6.1.4, no results were registered for this case. However the behavior that was observed during debugging is illustrated in figure 7.5 to give a general idea of how the system behaved. The traffic shifted from both AP's, to only one, and back to both. This went on and on as the AP's drifted back and forth in time relative to each other.

Given that there are no results stored, the numerical traffic values are not known, however given the design of the protocol the traffic should spit fairly evenly among the two available AP's, meaning that they should each observe the same traffic as if they had been alone with half the total amount of nodes. Based on this it is assumed that the traffic per AP during the common periods is equal to the traffic experienced in other tests with ten nodes on one AP. The periods with only one AP in sync should naturally have the same throughput as other tests with 20 nodes and one AP.

The proper execution of these tests is among the top priorities for future work, together with the implementation of a more accurate synchronization function.

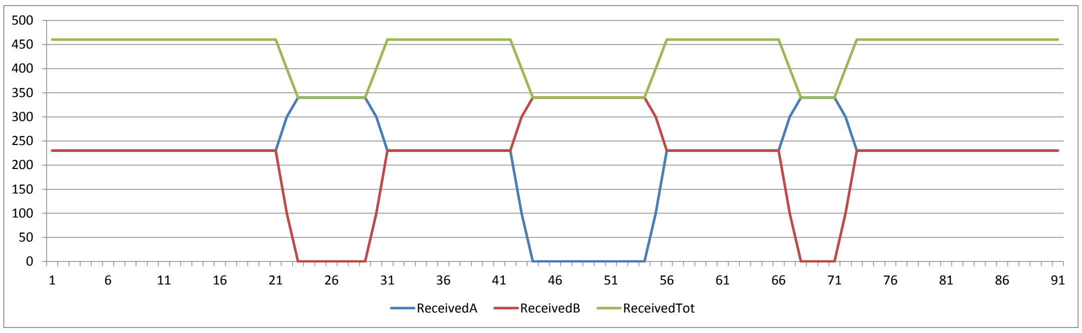


Figure 7.5.: Model of the behavior with two AP's, numerical values are assumptions based on other test results while the oscillating distribution of traffic is observed during debugging and testing.

7.7. Channel swap throughput tests

Description In order to determine the effect a channel swap has on the throughput a new test is conducted forcing the AP to swap channel at varying intervals. The internal channel swapping is disabled and all nodes are active throughout the whole test to give a continuous and constant traffic load.

The test program utilizes the channel swap function in the interface to initiate the transitions. This function allows the application to command the protocol to swap channel, but does not provide any influence on how the procedure is executed or to what channel the AP moves. The results are shown in figure 7.6.

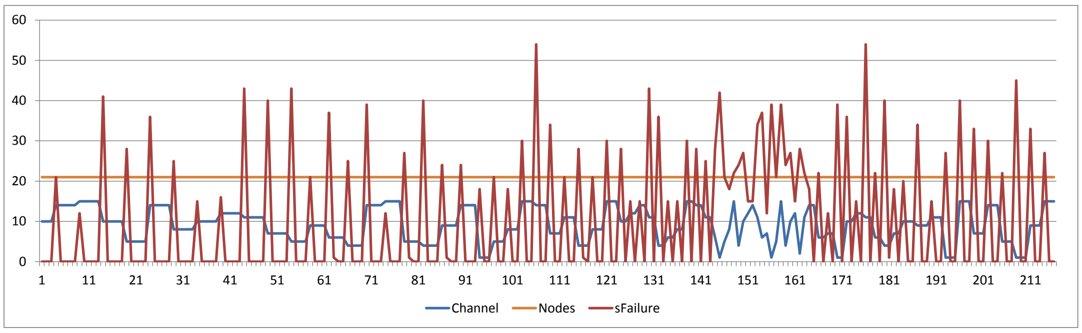
Channel swap issue As it turned out the interface function did not work as intended, causing a vast amount of disconnects and timing issues. This turned out to be due to concurrency issues as the interface function itself performed most of the analysis involved in the channel change without being atomic. Thus it would get interrupted by the protocol causing corrupted data and inconsistencies. As such this first test must be classified as a complete failure, but it revealed a serious issue that had to be taken care of.

Code 7.4: Test program: TestresultsChSwap21NodesNoSync1-21-05-13

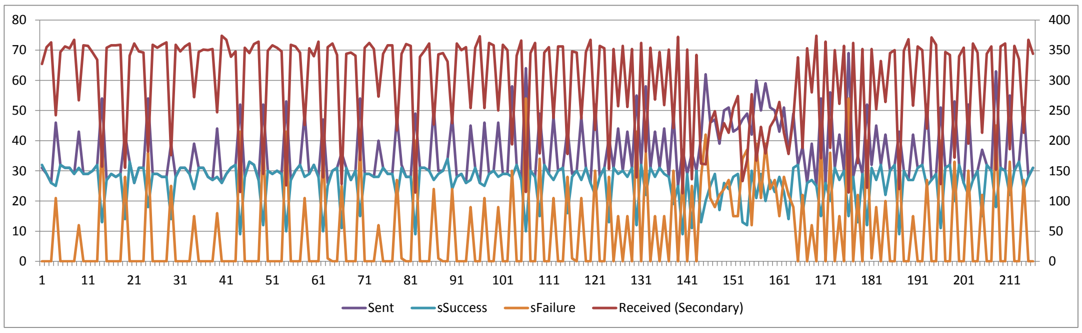
```
uint8_t cmdList[][8] =  {
    {2,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,5,0,0,0},
    {3,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC_NOCH,0,0,0},
    {5,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {4,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {3,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {2,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {1,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x20 - EDITED OUT FOR READABILITY --//
    {2,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {3,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {4,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {10,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},
    {5,255,CMD_RESET_AP,0,0,0,0,0},
    //MUST BE TERMINATED BY THE FOLLOWING ARRAY!
    {0,255,0,0,0,0,0,0}};
```

Node message size	AP message size	Node count
11 byte	8 byte	21
Test length	Channel Swapping	Synchronization
220 seconds	Manual	Disabled

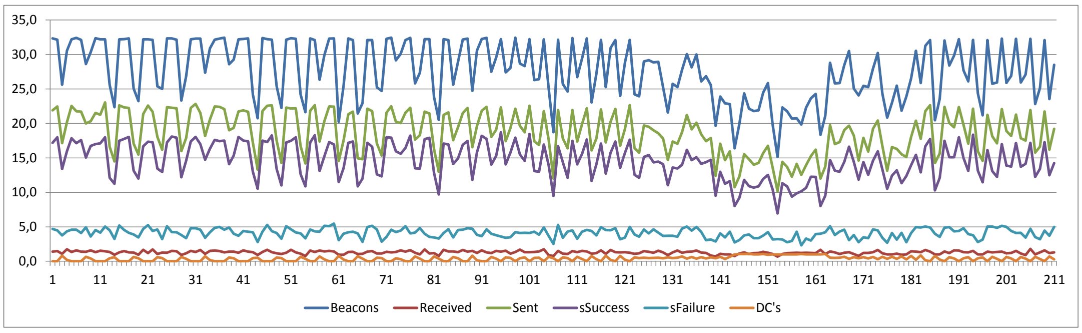
Table 7.4.: Major test parameters: TestresultsChSwap21NodesNoSync1-21-05-13



(a) APCH



(b) AP



(c) NODE

Figure 7.6.: TestresultsChSwap21NodesNoSync1-21-05-13

7.8. Channel swap throughput tests, after bug fix

Description After having the first channel swap test fail miserably the interface function was remade. In short the interface function itself was changed to only set a flag indicating to the protocol that it should initiate channel swapping. This flag is checked by the protocol the same place as it is checking the current channel quality estimation, reusing the triggered channel swap procedure. In total this means that a large code block, not doing what it was supposed to, was replaced with only a few lines of code doing exactly what it should. The test results can be seen in figure 7.7.

Minor artifacts In figure 7.7(b) the same tendency to an occasional failed transmission as in section 7.5 can be seen. According to the raw data the failures do not happen at the same time as the channel swaps, indicating that they are not connected. This is also backed up by the fact that the total amount of failed transmissions is approximately equal in the test without channel swapping as it is in this test.

The phenomenon is further discussed in section 8.5.

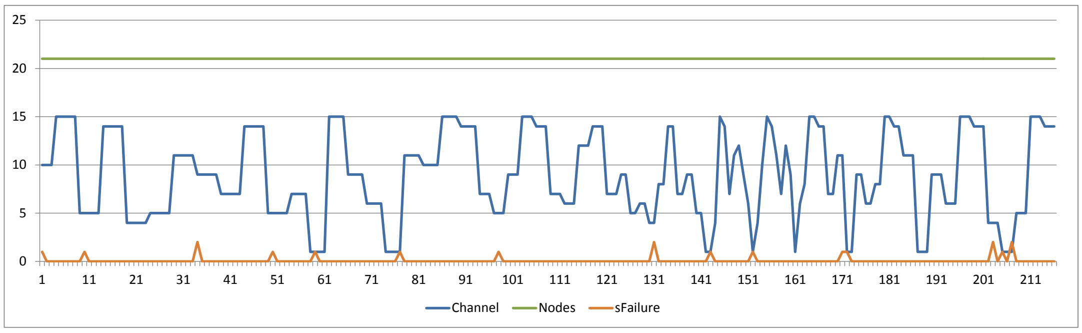
Channel swap penalty The results from this test, and an analysis of the performance penalty for swapping channels, are further discussed in section 8.4.

Code 7.5: Test program: TestresultsChSwap21Nodes3APChSwapChange-21-05-13

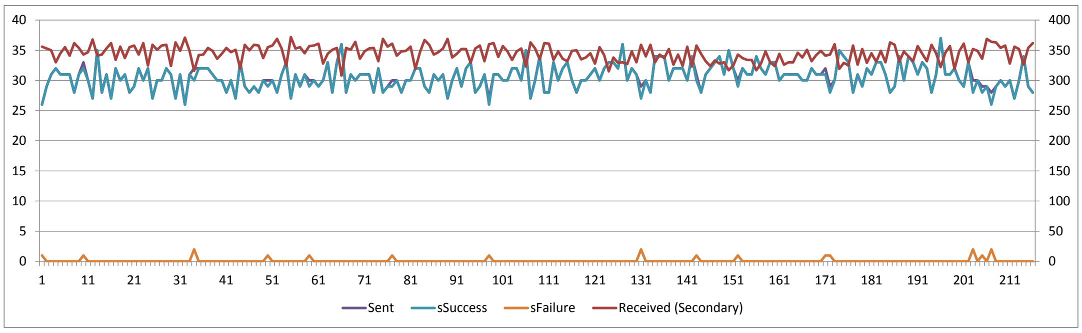
```
uint8_t cmdList[][8] =  {
    {2,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,5,0,0,0},
    {3,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC_NOCH,0,0,0},
    {5,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {4,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {3,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {2,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {1,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x20 - EDITED OUT FOR READABILITY --//
    {2,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {3,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {4,0,CMD_AP_CONFIG,'-',0,0,0,0}, //-- x10 - EDITED OUT FOR READABILITY --//
    {10,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},
    {5,255,CMD_RESET_AP,0,0,0,0,0},
    //MUST BE TERMINATED BY THE FOLLOWING ARRAY!
    {0,255,0,0,0,0,0,0}};
```

Node message size	AP message size	Node count
11 byte	8 byte	21
Test length	Channel Swapping	Synchronization
220 seconds	Manual	Disabled

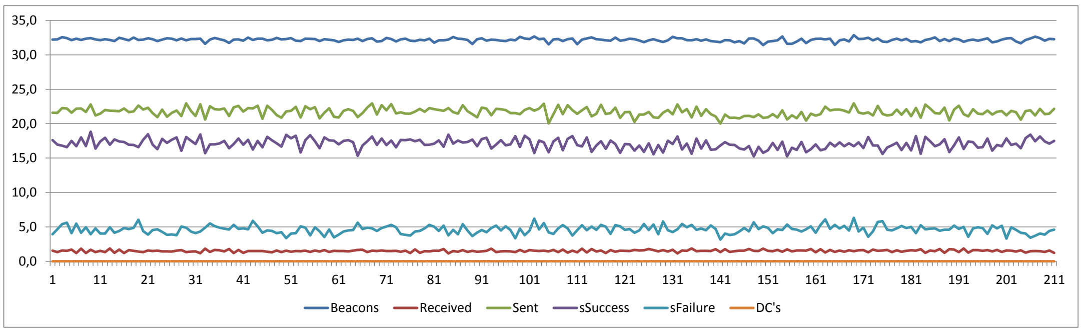
Table 7.5.: Major test parameters: TestresultsChSwap21Nodes3APChSwapChange-21-05-13



(a) APCH



(b) AP



(c) NODE

Figure 7.7.: TestresultsChSwap21Nodes3APChSwapChange-21-05-13

7.9. Channel footprint tests

Description With the core functionality proven to work nicely the channel footprint reported by the nodes has to be assessed. Given that the automatic channel quality assessment on the AP does not work as intended, as seen in section 7.4, the channel footprint is currently of limited use, but it will form the basis of any channel swaps initiated by the application.

The purpose of the tests is to see how the different filtering options appear, both on the nodes and on the AP. Generally the optimal solution is a filtering that gives the least changes in perceived quality over time, given that the environment does not change significantly. A good filtering will report channels with rare or sporadic noise as good choices for the AP, as a completely noise free channel is not particularly realistic.

Discarded setting The last setting in the test procedure turned out to be outside the parameters accepted by the nodes, thus the results from this particular run is discarded. In detail the nodes would only handle a "maximum factor" setting of 200, thus the attempt to apply the value 250 caused the filter to malfunction giving more or less random results.

Environment The environment during testing is considered very stable as the tests are conducted in a sparsely populated office environment. Two Wi-Fi channels are present, one with reasonable signal strength (-50dBm) in channel one, and another very weak (-87dBm) in channel three, a screenshot of the Wi-Fi measurements are shown in figure 7.8. There are also unidentified consistent noise on channel 13 and 16. This noise has been observed to be present during all tests and can as such be considered a constant interference. The measured noise in channel 16 is further discussed in section 8.6.

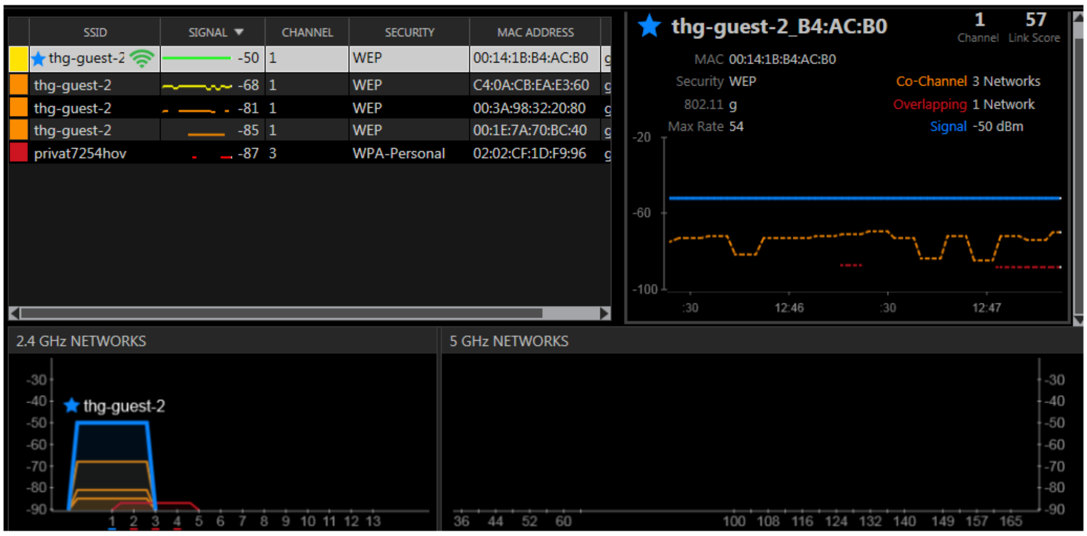
Multirun In order to give an as good as possible basis for deciding the optimal solution the tests are run two times one day apart. The AP is also located on different channels during the two runs to give a rough estimate of how big part of the signal footprint that is caused by crosstalk from the protocol itself.

Crosstalk In the first test the AP operates on channel four. In the second run it has been moved to channel 10. What can be observed in both tests is that the channel immediately following the active channel tends to have low energy readings on the nodes. This comes from the fact that with the given traffic load the AP will very rarely fit three messages in one beacon, meaning that the third acknowledge slot is generally not used. This means that there is effectively no crosstalk for the nodes to register through their energy detection procedure on this channel. The second following channel however, generally has consistent medium level readings. This makes perfect sense since one or more nodes are very likely to have gained access to the active channel at this point and are transmitting their data frames. This problem would to a large degree disappear when the nodes are moved further apart, making these readings somewhat artificial.

The crosstalk also comes to view when looking at AP's channel footprint. The channel immediately following the active channel can be seen to regularly turn deep red, indicating that the AP receives reports that the channel is actively used by another AP. Naturally this is not correct, but as described in section 7.1 this is caused by the nodes picking up and decoding the acknowledge frames transmitted on the active channel, thus interpreting the neighboring channel to be in active use.

A reasonable question to ask in connection to this is why there are little to no such crosstalk on the channels preceding the active one. Firstly the reason for this should be the nature of the protocol, as the nodes will make sure not to transmit so late that they run the risk of not completing in time. The energy measurement made by the nodes is performed during the last third of the subperiod, meaning that all nodes should effectively be blocked from transmitting during the energy detection phase on the last preceding channel. However, as described in section 7.2.3, there was also a bug in the protocol causing the nodes to be locked out of the channel several subperiods too early, with the test setup used here this would equate to one subperiod too many, meaning that the latest point at which a node could possibly be transmitting is around the middle of the second last subperiod before the beacon.

Result quality Generally it seems clear that these tests did not give a high quality result, mainly caused by the nodes being placed so close together that crosstalk had a large influence on the data. In fact, when looking at the raw data, this is very well illustrated by the one node that was connected to the computer, and thus located apart from the others. This node generally has lower energy readings and the readings on the crosstalk channels are virtually nonexistent. This aspect of the test is further discussed in section 8.7.



	SSID	SIGNAL	CHANNEL	SECURITY	MAC ADDRESS
🟨	thg-guest-2	-50	1	WEP	00:14:1B:B4:AC:B0
🟧	thg-guest-2	-68	1	WEP	C4:0A:CB:EA:E3:60
🟧	thg-guest-2	-81	1	WEP	00:3A:98:32:20:80
🟧	thg-guest-2	-85	1	WEP	00:1E:7A:70:BC:40
🟥	privat7254hov	-87	3	WPA-Personal	02:02:CF:1D:F9:96

thg-guest-2_B4:AC:B0 — 1 Channel, 57 Link Score

MAC 00:14:1B:B4:AC:B0
Security WEP
802.11 g
Max Rate 54
Co-Channel 3 Networks
Overlapping 1 Network
Signal -50 dBm

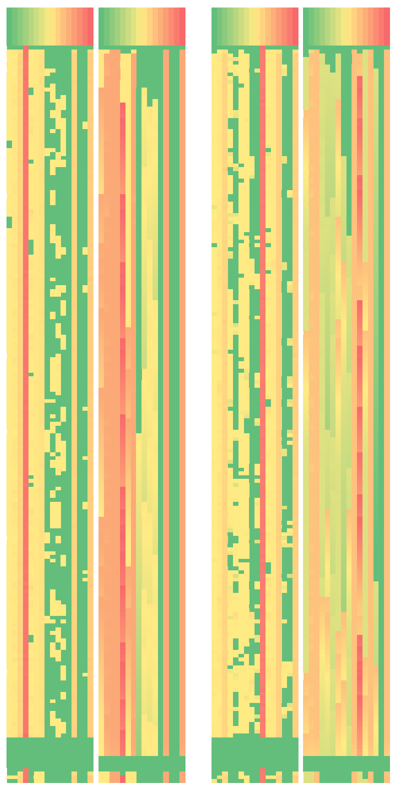
Figure 7.8.: Wifi channels at the execution of TestresultsChMgmt21Nodes15x200Sec2-21-05-13.

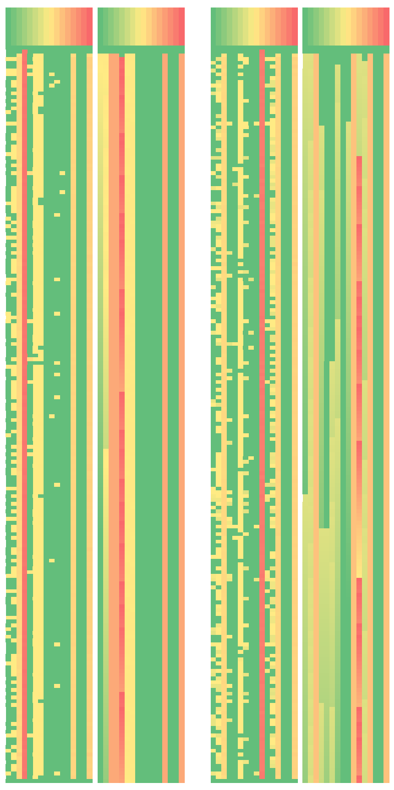
Code 7.6: Test program: TestresultsChMgmt21Nodes15x200Sec(1 / 2)-(20-05-13 / 21-05-13)

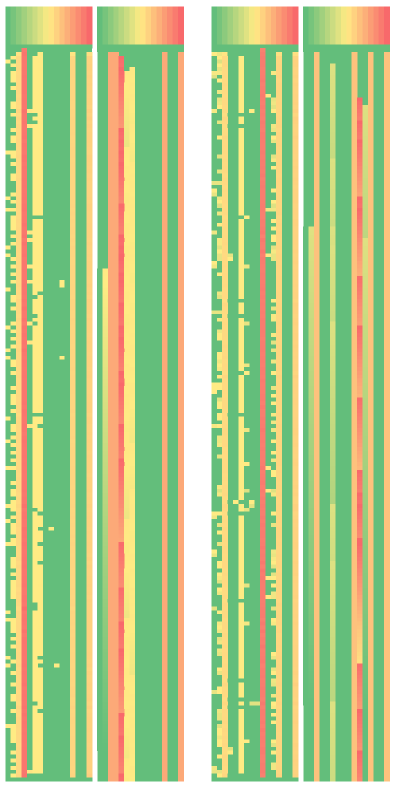
```
uint8_t cmdList[][8] =  {
                {3,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
                {2,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC,0,0,0},
/*Envelope*/    {185,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,0,0,0,0},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*Avg5*/        {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,2,5,0,0},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*Avg10*/       {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,2,10,0,0},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*Avg20*/       {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,2,20,0,0},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(10,0,50)*/   {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,10,0,50},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(5,0,50)*/    {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,5,0,50},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(10,0,100)*/  {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,10,0,100},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(5,0,100)*/   {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,5,0,100},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(10,5,50)*/   {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,10,5,50},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(5,5,50)*/    {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,5,5,50},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(10,10,100)*/ {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,10,10,100},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(5,10,100)*/  {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,5,10,100},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(10,0,150)*/  {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,10,0,150},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(10,0,200)*/  {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,10,0,200},
                {5,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,10,0,0,0},
/*(10,0,250)*/  {195,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_CH_MGMT,1,10,0,250},
                {10,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},
                //MUST BE TERMINATED BY THE FOLLOWING ARRAY!
                {0,255,0,0,0,0,0,0}};
```

Node message size	AP message size	Node count
26 byte	8 byte	21
Test length	Channel Swapping	Synchronization
15 x 200 seconds	Disabled	Disabled

Table 7.6.: Test parameters: TestresultsChMgmt21Nodes15x200Sec(1 / 2)-(20-05-13 / 21-05-13)


(a) 1.1 (b) 2.1


(c) 1.2 (d) 2.2


(e) 1.3 (f) 2.3

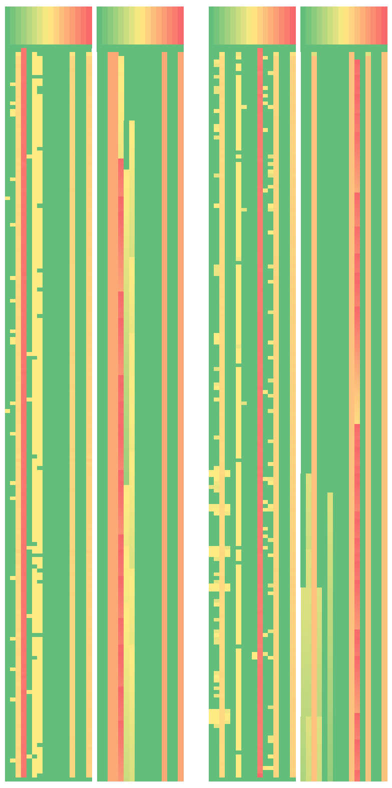
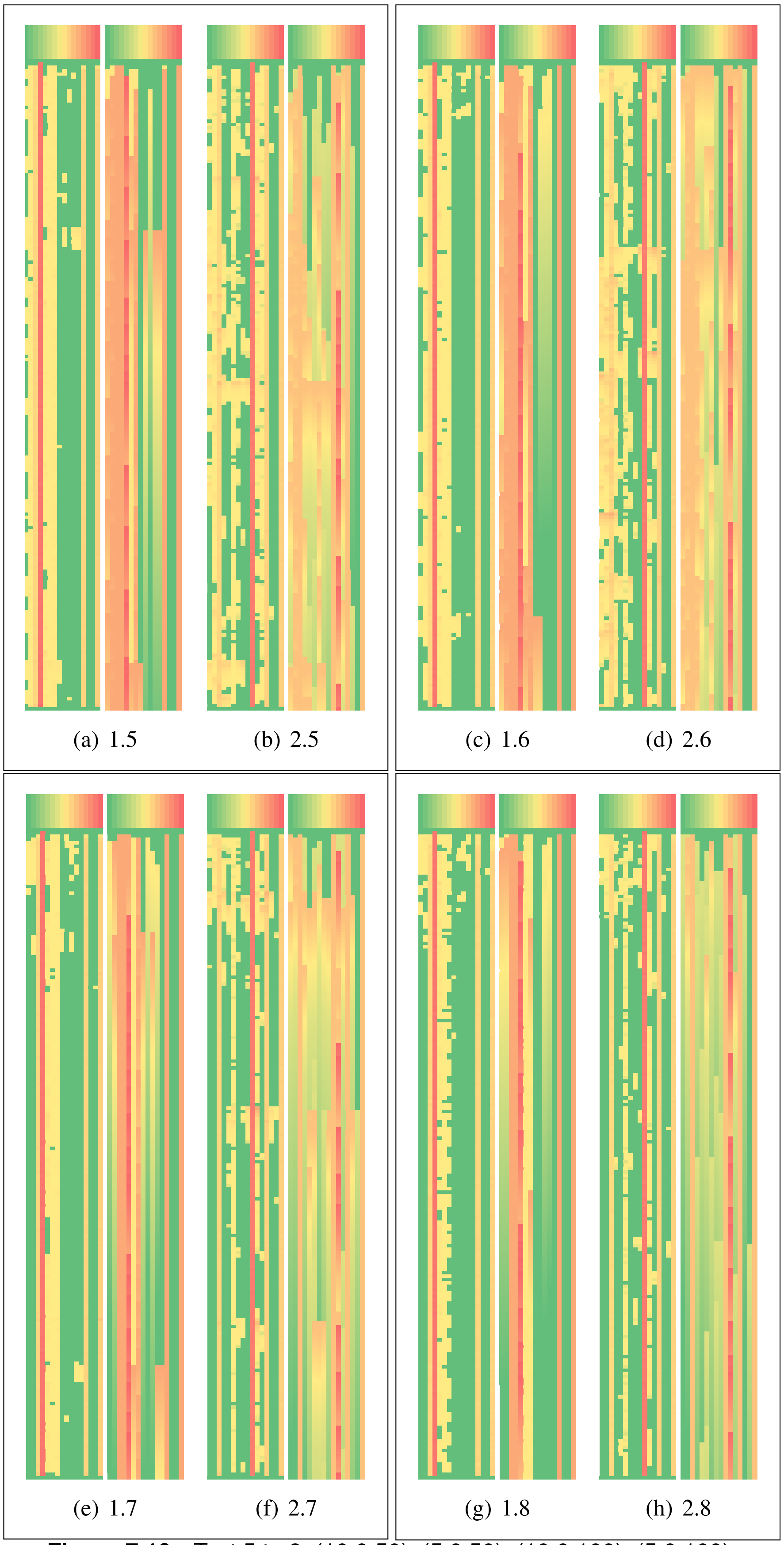

(g) 1.4 (h) 2.4

Figure 7.9.: Test 1 to 4, Envelope, Avg5, Avg10, Avg20.



(a) 1.5 (b) 2.5 (c) 1.6 (d) 2.6

(e) 1.7 (f) 2.7 (g) 1.8 (h) 2.8

Figure 7.10.: Test 5 to 8, (10,0,50), (5,0,50), (10,0,100), (5,0,100).

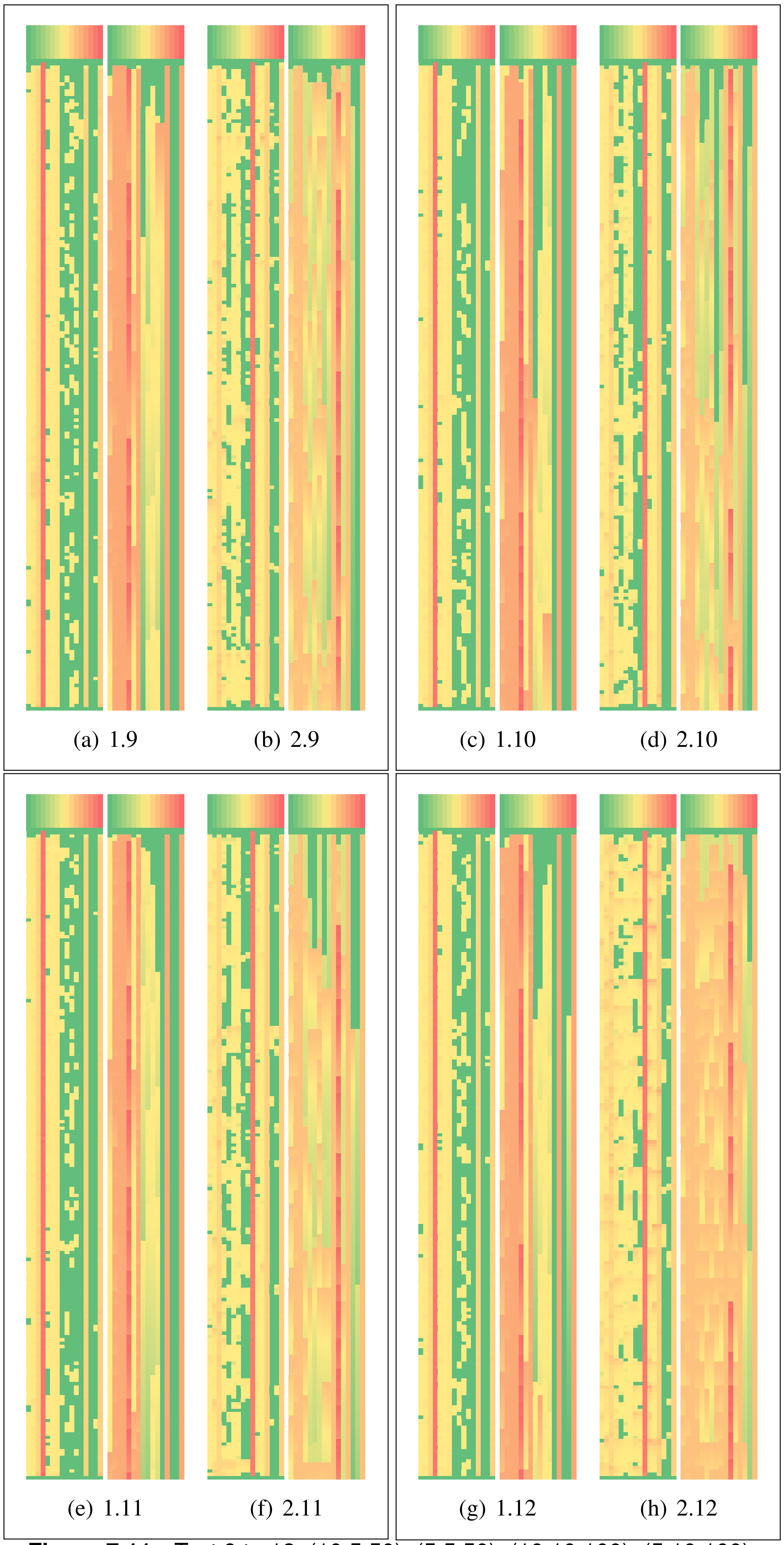
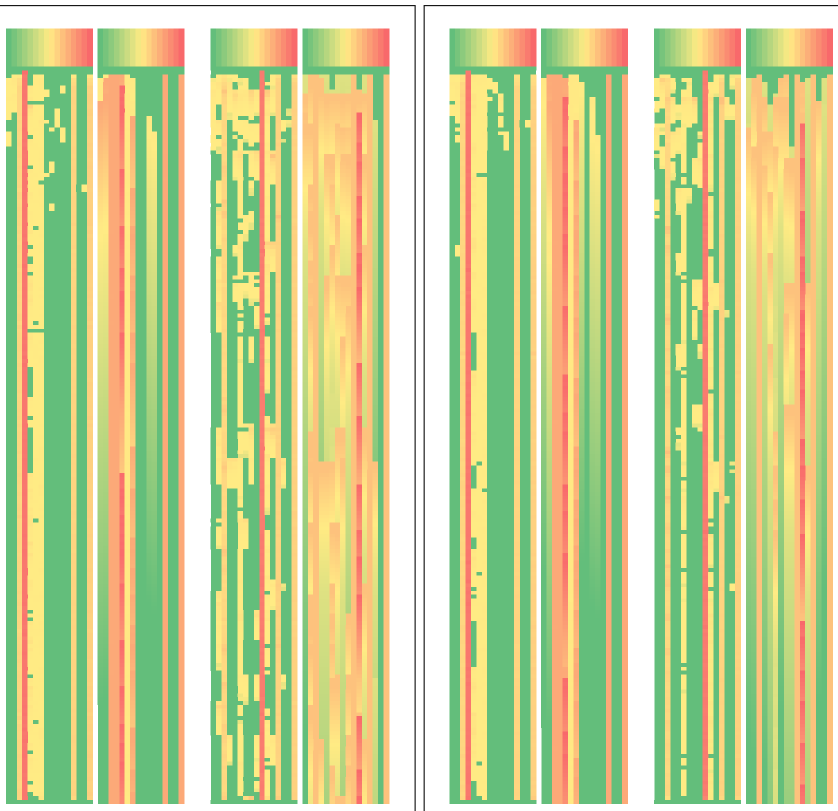


Figure 7.11.: Test 9 to 12, (10,5,50), (5,5,50), (10,10,100), (5,10,100).



(a) 1.13 (b) 2.13 (c) 1.14 (d) 2.14

Figure 7.12.: Test 13 and 14, (10,0,150), (10,0,200).

7.10. Single AP statistics tests

Description As a final test of the throughput and general performance of the protocol the maximum number of available nodes was used. This included using one of the AP's as the 22'nd node. With this setup the same test was run over and over a total of eight times. The results from all the tests were then averaged meaning that the resulting values will suppress most random or fluctuating factors.

The resulting values are plotted together with the theoretical values for the given scenario in figure 7.13.

Throughput dip One thing to note is the reported sudden decrease in transmissions to and from the nodes every ten seconds. The explanation for this is simply that once a new node is activated the first report from that node will contain measurements for anywhere between 0 and one second depending on the internal timer of the node. This means that these values are artificially low and should be ignored when analyzing the results.

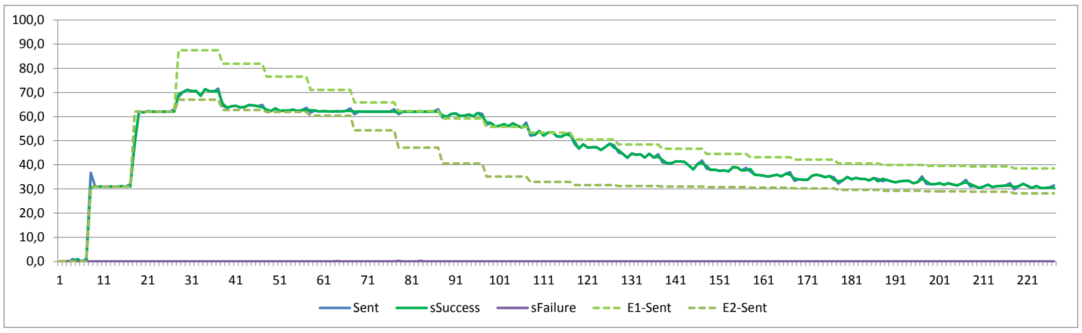
Extended measurement representations In order to compare the results to the specifications and ease the comparison with other protocols the results are also averaged and reformatted to represent packet loss percentage and throughput in bytes per second. Also a measure of the latency has been calculated, however since no dedicated measurement of latency has been made this number is calculated by averaging the time between two messages from a node. Still this gives a very good indication since the node will attempt to send the next message immediately after completing the previous transmission. These measures are presented in figure 7.14.

Code 7.7: Test program: Testresults22Nodes10secChStats(Multirun)-23.05.13

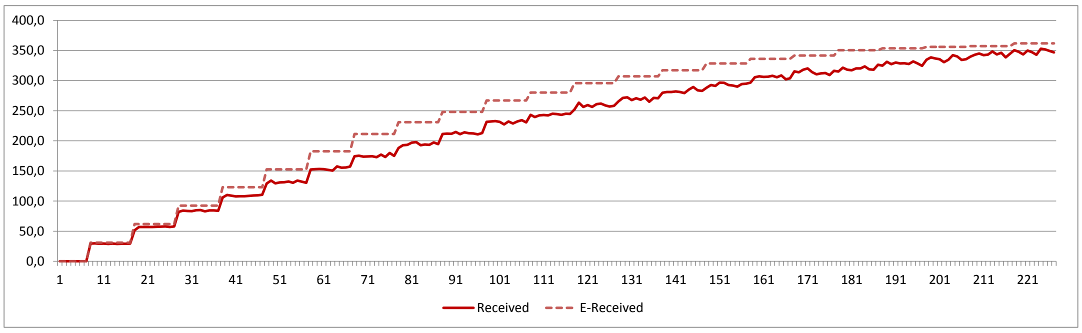
```
uint8_t cmdList[][8] =
            {{7,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},
            {3,0,CMD_NODE_CONFIG,NODE_CONFIG_SET_MODE,NODE_MODE_1SEC_CHSTAT,0,0,0},
            {10,1,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            {10,2,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            /* EDITED FOR READABILITY */
            {10,20,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            {10,21,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            {10,22,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,0,0,0,0},
            {10,0,CMD_NODE_CONFIG,NODE_CONFIG_SILENCE,255,0,0,0},

            {5,255,CMD_RESET_AP,0,0,0,0,0},
            //MUST BE TERMINATED BY THE FOLLOWING ARRAY!
            {0,255,0,0,0,0,0,0}};
```

Node message size	AP message size	Node count
14 byte	8 byte	1-22
Test length	Channel Swapping	Synchronization
8 x 230 seconds	Disabled	Disabled

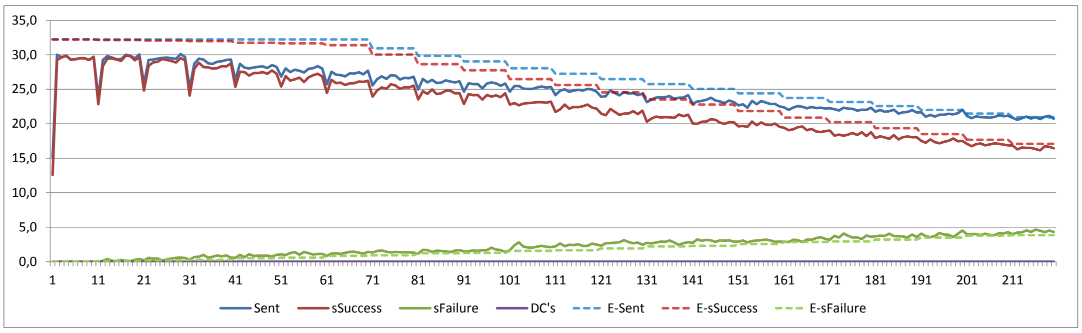
Table 7.7.: Major test parameters: Testresults22Nodes10secChStats(Multirun)-23.05.13



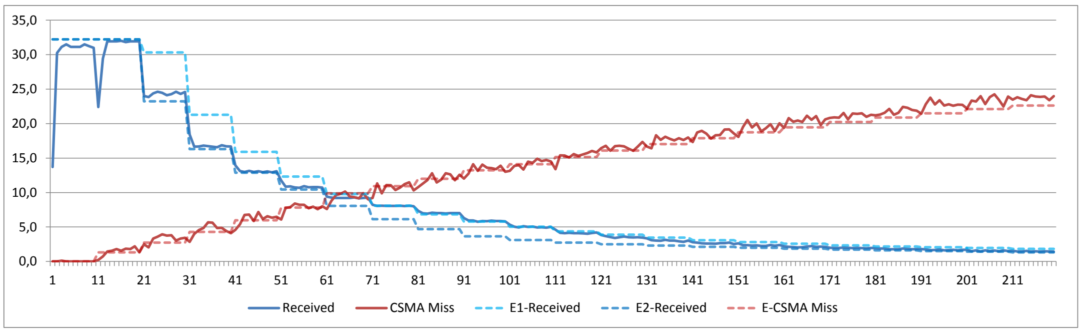
(a) AP Sending



(b) AP Receiving

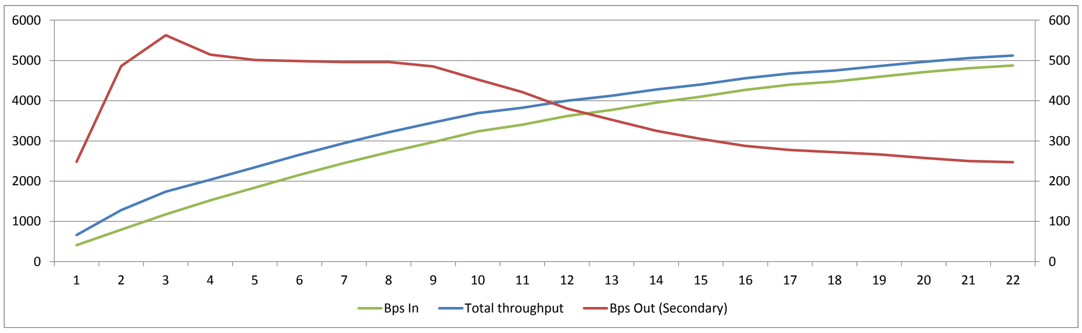


(c) Node Sending

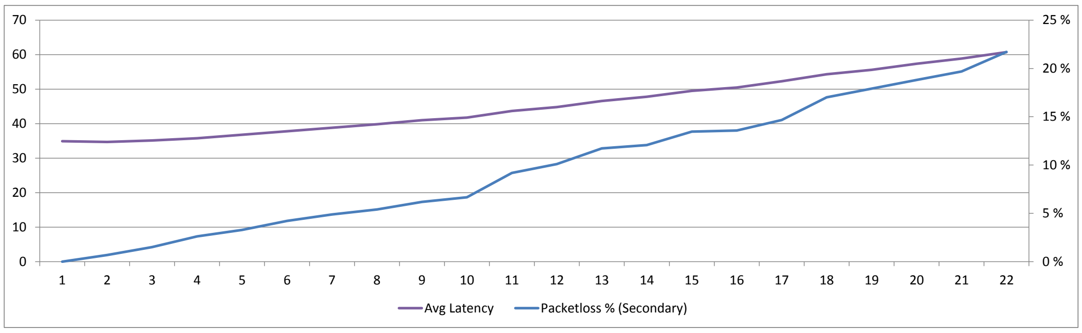


(d) Node Receiving

Figure 7.13.: Testresults22Nodes10secChStats(Multirun)-23.05.13



(a) Throughput in bytes per second.



(b) Latency and packet loss percentage.

Figure 7.14.: Test results converted to Bps throughput, packet loss percentage and latency in milliseconds.

8. Discussion

8.1. General results

Given the results in section 7.10 it is clear that the protocol does function according to the specifications when looking at one channel in isolation. The performance is as expected compared to the theoretical values and there are no glitches or errors causing instability.

The same is the case when performing channel swaps in section 7.8, though this test reveals a very small amount of transmission errors that should, theoretically, not have been present.

Still the successful results are mainly achieved once a lot of the critical functionality has been removed. In order for the protocol to be used in the target application it is an absolute must that the parts that currently do not work are corrected and tested properly.

8.2. Synchronization problems

The test in section 7.3 reveals the need for a more accurate synchronization function, and possibly other hidden errors connected to the synchronization. Given the inability to debug this part during implementation it was to some extent expected that there would appear some issues, but the severity of the problems was far worse than expected.

Still there is light at the end of the tunnel, namely the observed behavior when testing with two AP's during debugging prior to the hardware breakdown. This behavior is exactly as expected under the circumstances, and gives a strong indication that the fundamental concept is reasonable and that a well working system is achievable if only the accuracy are improved to a proper level.

Problem analysis The synchronization function has been the object of considerable rethinking following the test period. First of all, up till now, the timers were only set when the error between the expected timing and the received timing was sufficiently low. This was done to avoid setting the timings if the sync signal was delayed significantly by ongoing interrupts. However this approach is most likely the main reason why there are so many disconnects during the tests in section 7.3.

In figure 7.2(a) the dynamic offset of the PLL is seen to oscillate around 3-4 indicating that the sync signal has around 3,5 symbols error compared to the correct 1Hz. Given $16\mu s$ per symbol this means a frequency of approximately 0,999944Hz. Given the general accuracy of microcontroller crystals this has to be an acceptable error, and this error should be handled by the PLL. In addition to this comes the 4 symbols offset due to rounding errors when calculating the period and subperiod timings as explained in section 3.9.2. In total this means that the timers should be delayed 7-8 symbols once each second. This is no problem for the nodes as they will adjust their timings accordingly, so long as all AP's perform approximately the same delay and stay within a distance from each other no bigger than what is handled by the timing buffer described in section 2.5.6.

In the implementation however, this failed. When an AP had its sync signal delayed by an ongoing interrupt it would not adjust its timers at all that second. This caused the timers to be 7-8 symbols in front of the correct, global timing. This error would then accumulate each time the AP had its signal delayed, and each time the offset of the PLL oscillated too far away from the correct value. If this happened for five consecutive seconds the error would become almost 40 symbols, if it went on for ten seconds the error would build up to almost 80 symbols. With a subperiod timing of 126 symbols, as stated in section 3.9.2, the accumulated error would equate to a full subperiod after only 15 seconds. When the AP then finally adjusts the timers, the beacon will suddenly move out of its subperiod, making the nodes unable to receive it, thus they all disconnects.

Immediate solution The first and most obvious solution to this is to set the timers each second regardless of any delays, but instead of setting them with static values as is done now, the value is calculated dynamically based on the estimated delay of the sync signal. This way, if the sync signal is delayed some 20 symbols by an ongoing interrupt all timers will be set with their static value minus 20. This should then completely remove the error accumulation described above.

Given the analysis of the problems it is believed that this should stabilize the system with a single AP. Since the beacon will never suddenly jump, as it did with the current implementation. This should lead to the tests in section 7.3 giving stable and uninterrupted results.

Further improvements Still some issues remain for the sync between multiple AP's. Namely the fact that the PLL will drift and oscillate around the desired value, and this could very well lead to the AP's drifting further apart than the accepted timing buffer on the nodes.

To minimize this error one possible solution would be to increase the synchronization frequency. This will give the PLL more samples meaning that it would be able to correct the erroneous drifting faster. Theoretically this should make the oscillations smaller but more frequent. Since only the amplitude matters in this case this is likely to improve the result significantly.

In connection to this some adjustments could also be made to the PLL itself. Firstly it could be made very slow, effectively making it into a low pass filter for the sync signal. This way it should be able to correctly adjust to the close to constant error caused by an inaccurate sync frequency, while the fluctuating errors from interrupt delays would to some extent be ignored. Secondly it would be possible to exploit the fact that all non-constant errors will only cause the sync signal to be delayed, never to trigger prematurely. This means that the PLL could treat the signal different if it arrives too late or too early. If the sync signal arrives before it was expected it has to mean that the expected value is too large, indicating a too large factor, thus the PLL could adjust the

Code 8.1: Pseudo code for a new and improved synchronization function.

```
sync(received){
  offset = expected - received;
  if(offset < 0){ //Delayed
    factor += SmallValue;
  }else if(offset > 0){ //Premature
    factor -= offset;
  }
  SetTimers(Const + offset);
  expected += SyncPeriod + factor;
}
```

factor quite abruptly without any risk. If the signal is delayed however, it is most likely caused by interrupts, meaning that the PLL should only perform a minor adjustment of the factor.

In code 8.1 a simple pseudo code for a new sync function is presented. Here it is shown how the factor is adjusted significantly if the sync signal arrives prematurely. When the signal is delayed however the factor is only adjusted slightly. Also the timers are always set, but the offset between the expected sync and the received sync is added to compensate. This improved function, in combination with an increased sync frequency, should provide a far better theoretical solution than what is currently in place.

8.3. AP Channel assessment errors

The tests in section 7.4 reveals that the AP incorrectly assess the channel to be noisy as the number of nodes increase. The reason for it is described in section 6.1.6 as the detection of corrupted frames.

The way the channel assessment works is by subtracting a set, or random, value from the quality indicator each time an undesired event happens. Each beacon trigger the quality indicator is incremented, meaning that if no undesired events occur the indicator will stay at its maximum value. Different events cause different subtractions, with the most noticeable event being the reception of a valid frame with a different network ID. This means that two AP's are competing on the same channel, and it is desired that one of them swaps as soon as possible. The reception of a corrupted frame is interpreted as the channel being noisy, and causes a minor decrementation. Lastly the detection of a higher than desired energy level will cause the smallest subtraction.

Problem analysis The reason for the channel assessment failing as it did is due to the collisions caused by the CSMA/CA procedure. As the number of nodes increase the number of collisions will also increase, and this is very well reflected in the test results in section 7.4, as the channel quality indicator is clearly decremented faster and faster as the number of nodes increases.

Possible solution An immediate solution could be to simply disable the decrementation caused by the corrupted frames, and that could be a reasonable temporary fix. Still a better and more permanent solution is for the AP to measure the channel throughput and modify the impact a corrupted frame has on the quality estimator based on this. This can be achieved by having the AP count the number of frames received over one subperiod and use the resulting number as a basis for the channel quality decrementation, through some conversion method. The simplest solution would be to hardcode an array saying that when receiving X messages per period the first Y corrupted messages shall be ignored. The X-Y conversions can be derived directly from the theoretical calculations in chapter 5, or the practical results obtained in section 7.10. Given the random nature of CSMA/CA the number of legitimate collisions has to be overestimated slightly, or the AP could average the numbers over several periods if the memory footprint and overhead connected to that functionality is acceptable.

Further improvements Another situation that is not currently very well covered by the channel assessment is if two AP's operate on the same channel and in synchronization. This will cause them both to transmit in parallel, meaning that neither will detect the other's beacon. A way of detecting this event is to evaluate the acknowledgements received. Generally the acknowledgements should be very stable and rarely get corrupted or in other ways go missing. The things an AP should listen for are the following; firstly the reception of an acknowledge frame without sending any messages is a clear indicator of an AP collision. Secondly comes the reception of too many acknowledgements, and thirdly the reception of valid acknowledgements from node ID's that should not have received a message in the preceding beacon. All these artifacts can happen if two AP's transmit their beacons on top of each other, but it all depends on whether the nodes are able to decode the beacons or not. A just as likely outcome is that neither AP receives any acknowledgements at all, since the nodes were not able to correctly decode the frames. This case gives a slightly less clear indicator since acknowledgements can go missing if the nodes have moved out of range, or if the node is currently transmitting on a different channel, as shown in figure 2.4(d). This means that the AP cannot immediately assume a collision if a single acknowledge goes missing, if two or more acknowledgements disappears however, the probability of a collision is greatly increased and the impact on the quality estimator should be significant. A reasonable solution would be to set three limits, one for each missing acknowledgement, and generate a random number modulo each limit when the first, second and third acknowledge goes missing. The limits could very well be something like 2, 16 and 64, meaning that three missing acknowledgements will have the same impact on the estimator as the reception of a frame with a different network ID.

8.4. Channel swap penalty

As stated in section 3.5.6 the AP will move to a new channel immediately after the acknowledge phase, leading to all incoming data frames in the following access window being lost, as they are transmitted on the wrong channel. In addition to this it is stated in section 3.7.7 and table 3.3 that a channel will be marked as unstable by the nodes until two consecutive beacons has been received. This means that the new channel that the AP has moved to will not be used for transmission by the nodes for one additional subperiod. In total this results in the loss of two subperiods of transmissions. With

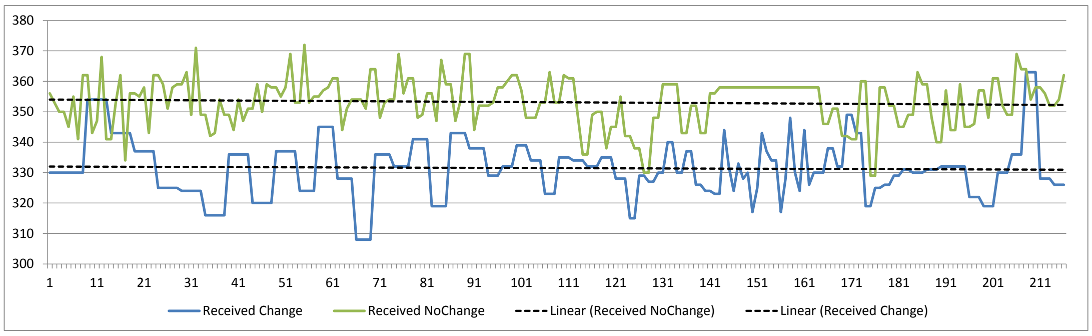


Figure 8.1.: Estimation of channel swap penalty based on test results.

31 beacons per second this equates to $6,45\%$ of the total throughput capacity over one second.

In section 7.8 a test is conducted where the AP is forced to swap channels very frequently. In order to measure the impact a channel swap has on the throughput, the results have been filtered into two separate graphs. The "Received Change" graph contains the values reported during the seconds where the AP has performed a channel swap. The "Received NoChange" line represents the reports from seconds where the system were stationary on one channel. In order to make the graphs continuous the value from the previous cell is simply repeated in cells where the specific graph should filter out the reported value. In the resulting plot the linear trend line for each graph is added. The final plot is seen in figure 8.1.

When averaging the filtered values, excluding the repeated cells, the average number of frames during "channel swap seconds" are $331,6$, while the average during "stationary seconds" are $353,4$. The difference is $331,6/353,4 = 0,938$, meaning that the experienced penalty during this particular test is $6,2\%$. Given that these numbers are based on a single test it is reasonable to assume that there will be some random factors influencing the results. This can also be seen in figure 8.1, as the two filtered plots are fairly volatile. Still the results are so clear that it is reasonable to conclude that the behavior is exactly as expected.

8.5. Failed AP-transmissions

In the tests in section 7.5 and 7.8 it can be observed that the AP occasionally experiences a failed transmission. The reason for this is not known exactly, and the output from the test application does not give any further indications as to where the error is located. Still one interesting thing can be seen if the test results in section 7.10 are analyzed in detail. In this test there are virtually no failed transmissions at all. During the full eight test runs there are only a total of five failed AP-transmissions.

In between these tests one seemingly insignificant change was made to the node protocol code in an attempt to repair the error described in section 6.1.7. Namely to swap the interrupt routines connected to the symbol counter comparator two and three, these two routines are the acknowledge transmission trigger and the data frame transmission trigger. Logically speaking this change should not matter whatsoever, but it appears as if the modification did cause a change in behavior that had an impact on the occasional failed transmissions. This naturally also indicates that the original error was connected to the node not transmitting the acknowledgement correctly, rather than the AP not being able to receive it or the node not receiving the beacon.

Given that the problem seems to have been largely removed, the few failures during the test in section 7.10 could very well be caused by a noise spike or other random events, the conclusion is that no further actions are needed on this matter.

8.6. Channel 16 noise

During all tests, including the channel footprint tests in section 7.9, there has been detected a significant energy level on channel 16. The source has been unknown, but assumed to be a separate network of some kind. However during the analysis of the results something came to view. During the channel footprint tests one node was located slightly apart from the others. Node 21 was connected to the computer with JTAG to enable debugging, and was thus located both closer to the AP, and apart from the other nodes. The setup is shown in figure 7.1.

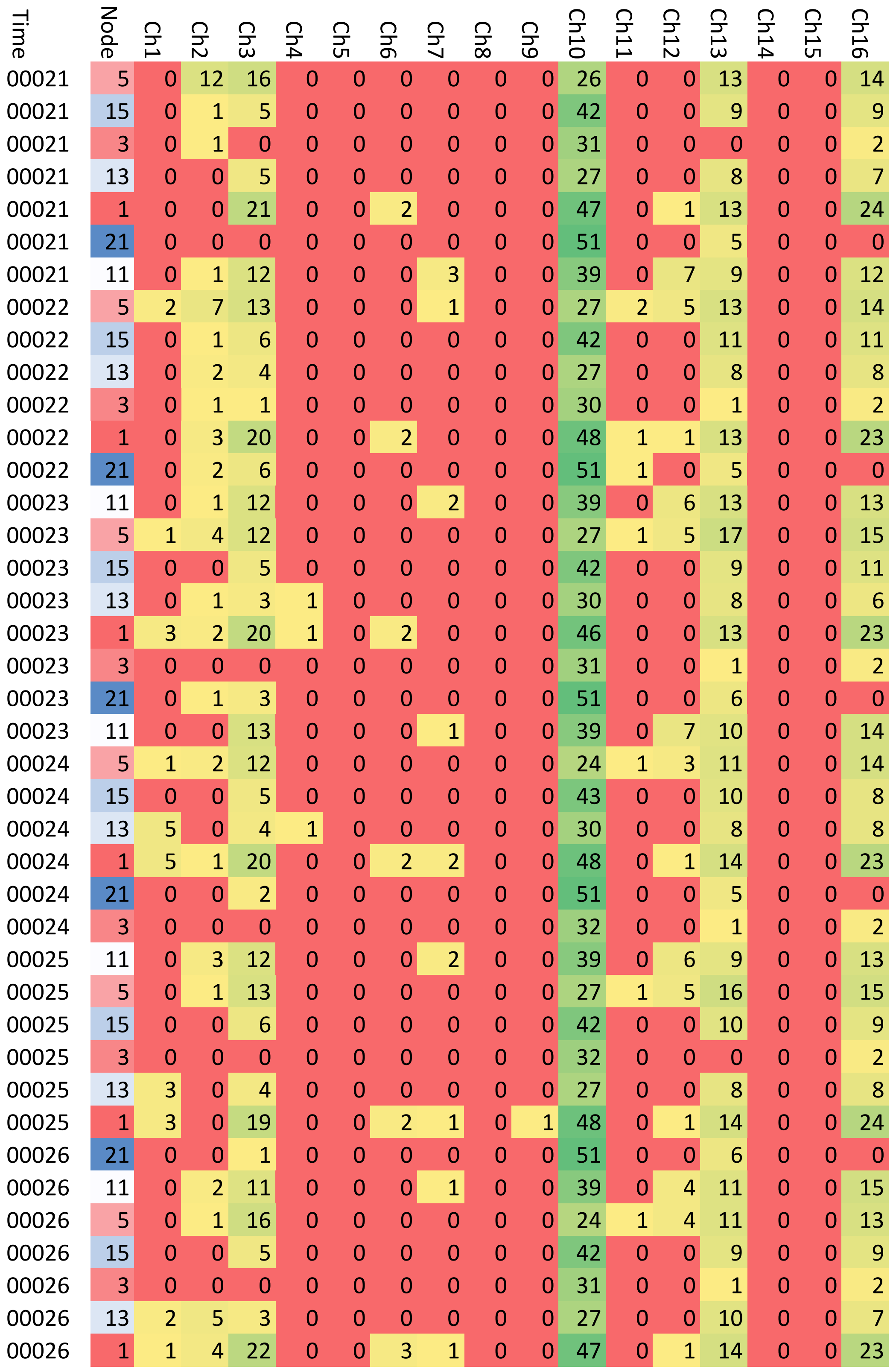
Problem analysis When looking at the measurements reported by the different nodes node 21 has quite different readings from the others. In figure 8.2 some samples are show, for node 21 and a small collection of other nodes to illustrate the issues.

Node 21 consistently reports zero energy on channel 16, while the rest of the nodes report readings ranging from 1 to 24. This could indicate that the source might not be an external network after all. The noise might be some sort of interference generated by the nodes themselves or by the power supply.

During the testing the nodes has been located very close together, and in many cases the antenna plug of one module are located only centimeters from either the microcontroller or the antenna plug of another module. It must be assumed that this can make them interfere in unpredictable ways, and thus that this might be the source of the strange readings.

The data currently available are too scarce to draw any conclusions, but the fact that the measurements are very consistent for each separate node, while the difference between the nodes are significant, indicates that the source is something static, and the fact that the noise has been observed also when debugging the protocol at a different geographical location, while still having the available nodes placed in immediate proximity of each other, supports this theory.

Possible further actions One interesting test in order to investigate this issue would be to repeat the exact same test with the exact same physical layout, and then try to move some nodes slightly to see if this changes the energy measurements. Also more nodes should be moved away from the current node-cluster to verify that the zero measurement given by node 21 is not simply an error.



Time	Node	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6	Ch7	Ch8	Ch9	Ch10	Ch11	Ch12	Ch13	Ch14	Ch15	Ch16
00021	5	0	12	16	0	0	0	0	0	0	26	0	0	13	0	0	14
00021	15	0	1	5	0	0	0	0	0	0	42	0	0	9	0	0	9
00021	3	0	1	0	0	0	0	0	0	0	31	0	0	0	0	0	2
00021	13	0	0	5	0	0	0	0	0	0	27	0	0	8	0	0	7
00021	1	0	0	21	0	0	2	0	0	0	47	0	1	13	0	0	24
00021	21	0	0	0	0	0	0	0	0	0	51	0	0	5	0	0	0
00021	11	0	1	12	0	0	0	3	0	0	39	0	7	9	0	0	12
00022	5	2	7	13	0	0	0	1	0	0	27	2	5	13	0	0	14
00022	15	0	1	6	0	0	0	0	0	0	42	0	0	11	0	0	11
00022	13	0	2	4	0	0	0	0	0	0	27	0	0	8	0	0	8
00022	3	0	1	1	0	0	0	0	0	0	30	0	0	1	0	0	2
00022	1	0	3	20	0	0	2	0	0	0	48	1	1	13	0	0	23
00022	21	0	2	6	0	0	0	0	0	0	51	1	0	5	0	0	0
00023	11	0	1	12	0	0	0	2	0	0	39	0	6	13	0	0	13
00023	5	1	4	12	0	0	0	0	0	0	27	1	5	17	0	0	15
00023	15	0	0	5	0	0	0	0	0	0	42	0	0	9	0	0	11
00023	13	0	1	3	1	0	0	0	0	0	30	0	0	8	0	0	6
00023	1	3	2	20	1	0	2	0	0	0	46	0	0	13	0	0	23
00023	3	0	0	0	0	0	0	0	0	0	31	0	0	1	0	0	2
00023	21	0	1	3	0	0	0	0	0	0	51	0	0	6	0	0	0
00023	11	0	0	13	0	0	0	1	0	0	39	0	7	10	0	0	14
00024	5	1	2	12	0	0	0	0	0	0	24	1	3	11	0	0	14
00024	15	0	0	5	0	0	0	0	0	0	43	0	0	10	0	0	8
00024	13	5	0	4	1	0	0	0	0	0	30	0	0	8	0	0	8
00024	1	5	1	20	0	0	2	2	0	0	48	0	1	14	0	0	23
00024	21	0	0	2	0	0	0	0	0	0	51	0	0	5	0	0	0
00024	3	0	0	0	0	0	0	0	0	0	32	0	0	1	0	0	2
00025	11	0	3	12	0	0	0	2	0	0	39	0	6	9	0	0	13
00025	5	0	1	13	0	0	0	0	0	0	27	1	5	16	0	0	15
00025	15	0	0	6	0	0	0	0	0	0	42	0	0	10	0	0	9
00025	3	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	2
00025	13	3	0	4	0	0	0	0	0	0	27	0	0	8	0	0	8
00025	1	3	0	19	0	0	2	1	0	1	48	0	1	14	0	0	24
00026	21	0	0	1	0	0	0	0	0	0	51	0	0	6	0	0	0
00026	11	0	2	11	0	0	0	1	0	0	39	0	4	11	0	0	15
00026	5	0	1	16	0	0	0	0	0	0	24	1	4	11	0	0	13
00026	15	0	0	5	0	0	0	0	0	0	42	0	0	9	0	0	9
00026	3	0	0	0	0	0	0	0	0	0	31	0	0	1	0	0	2
00026	13	2	5	3	0	0	0	0	0	0	27	0	0	10	0	0	7
00026	1	1	4	22	0	0	3	1	0	0	47	0	1	14	0	0	23

Figure 8.2.: Collection of individual samples from channel footprint test. Included maximum, node 1, minimum, node 3, separate node, node 21, and other nodes with similar last digit. As well as node 5 and 15 for more samples in-between.

8.7. Channel Footprint test

Problem analysis As stated in section 8.6 the noise registered in channel 16 appears to be some sort of local interference rather than an actual competing network. In addition to this the readings shown in figure 8.2 indicates that the crosstalk portion of the energy readings are significant for the nodes located closely together. While node 21, located close to a meter away from the other nodes, have very clean an consistent measurements, the nodes located in the cluster have multiple small readings in channels that should presumably be clear.

This is also reflected in figure 8.3 where the measurements for the full duration of the test are averaged, this means all values collected across all different channel assessment algorithms, close to one hour of testing. This figure is made to compare the values between the nodes over a period where they have been subject to the same test procedure and the same environment. Given the very long duration of the channel footprint tests, around 50 minutes, it can be assumed that any random differences have evened out leaving only the hardware differences and the local signal maxima and minima.

Figure 8.3 show that there are large individual differences between the modules. Some nodes appear to be extremely sensitive while others appear to experience signal losses. The most noticeable are node 1 and 3, where node 1 has by far the highest detected energy level on inactive channels while node 3 hardly picks up any energy at all. Interestingly however, neither have the most extreme readings on the active channel, though they both rank in the top/bottom 5.

In order to save some time and unpacking no nodes had their antennas mounted during any of the performed tests, meaning that they all operate solely with their antenna plug as an antenna. Given that the distance from the nodes to the AP is very short this should have no influence on the protocol functionality, since the signals are easily transmittable over this short distance anyways. But it is obvious that this was not a good choice for the tests involving features of the physical layer, like the energy measurements. It must be assumed that the antenna plugs have been subject to limited tuning and customization to make them good transmitters/receivers, resulting in widely different transmission properties.

Another thing to note in figure 8.3 is how the average energy level in total is higher in test two than in test one. This is well illustrated by looking at the color shade of the whole figure, where test two have a generally more yellow shade than test one, meaning the majority of the values are higher.

The exact source of this noise is unknown, but given that the tests were performed in an office environment it can simply be that there were more staff present during test two than test one, meaning more Wi-Fi traffic and generally more noise from electrical office equipment.

Possible further actions The different assessment algorithms can to some extent be compared using the data gathered, as the source of the registered noise is of no interest to the algorithm, and in the realistic scenario the nodes will be expected to observe different levels of noise depending on their physical location. Thus the tests are usable for determining what algorithms to further evaluate, but they do not represent the energy footprint an individual node will experience once properly assembled.

In order to get a better test on this matter the nodes should be assembled fully and installed in an active installation. This will remove most of the issues presumably experienced in this test, the nodes will be far apart, they will all have the same antenna gain, at least closer than what was the case in these tests, and they will be moving, meaning that local maxima and minima of interference will even out over the duration of the tests.

Node	Ch1	Ch2	Ch3	Ch5	Ch6	Ch7	Ch8	Ch9	Ch10	Ch11	Ch12	Ch13	Ch14	Ch15	Ch16	Sum	Ch4
1	0,3	1,4	20,2	1,2	2,6	2,4	0,0	0,1	0,2	0,0	0,0	13,7	0,0	0,0	22,0	64,1	47,0
2	0,0	0,1	10,7	0,2	0,0	0,1	0,0	0,0	0,0	0,0	0,0	5,7	0,0	0,0	10,0	26,8	31,9
3	0,0	0,3	0,4	1,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	1,1	3,8	33,0
4	0,1	1,3	6,9	0,1	0,0	0,5	0,0	0,0	0,0	0,0	0,0	6,6	0,0	0,0	7,4	22,9	45,1
5	0,4	1,4	12,8	0,3	1,0	2,4	0,0	0,0	0,0	0,0	0,0	11,6	0,0	0,0	13,2	43,3	25,2
6	0,6	1,6	12,6	1,4	1,3	2,4	0,0	0,0	0,0	0,0	0,0	9,6	0,0	0,0	14,3	43,9	45,0
7	0,0	0,3	8,6	2,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0	5,9	0,0	0,0	10,4	27,4	48,0
8	0,1	1,3	8,9	0,5	0,2	0,5	0,0	0,0	0,2	0,0	0,0	6,6	0,0	0,0	11,1	29,3	34,9
9	0,1	0,6	6,5	0,6	0,3	0,6	0,0	0,0	0,1	0,0	0,0	12,1	0,0	0,0	8,7	29,4	48,0
10	0,1	0,4	3,5	1,0	0,0	0,8	0,0	0,0	0,1	0,0	0,0	7,7	0,0	0,0	5,7	19,3	40,6
11	0,2	0,6	11,5	0,5	1,2	1,7	0,0	0,0	0,0	0,0	0,0	6,9	0,0	0,0	13,0	35,7	42,5
12	0,2	0,5	7,6	0,5	1,0	1,6	0,0	0,0	0,0	0,0	0,0	3,4	0,0	0,0	10,6	25,5	40,9
13	0,0	0,4	2,7	0,7	0,1	0,6	0,0	0,0	0,0	0,0	0,0	8,5	0,0	0,0	6,7	19,9	26,2
14	0,2	0,5	4,8	0,6	0,6	1,6	0,0	0,0	0,0	0,0	0,0	6,7	0,0	0,0	8,4	23,4	45,0
15	0,0	0,3	5,5	1,0	0,0	0,3	0,0	0,0	0,0	0,0	0,0	8,0	0,0	0,0	9,9	25,1	42,0
16	0,1	0,9	2,7	0,4	0,0	0,4	0,0	0,0	0,0	0,0	0,0	8,0	0,0	0,0	7,4	20,1	47,9
17	0,1	1,0	6,2	1,0	0,0	0,6	0,0	0,1	0,2	0,2	0,0	9,2	0,0	0,0	7,2	25,9	36,9
18	0,1	0,6	2,2	0,3	0,0	0,6	0,0	0,0	0,2	0,1	0,0	8,8	0,0	0,0	3,8	16,6	27,0
19	0,1	0,7	6,6	0,7	0,4	0,9	0,0	0,1	0,1	0,0	0,0	10,0	0,0	0,0	5,4	25,1	35,1
20	0,2	1,0	5,4	1,5	0,4	1,3	0,0	0,0	0,0	0,0	0,0	8,9	0,0	0,0	10,6	29,3	38,5
21	0,0	0,2	0,2	0,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	4,9	0,0	0,0	0,0	6,3	49,4

(a) Test1

Node	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6	Ch7	Ch8	Ch9	Ch11	Ch12	Ch13	Ch14	Ch15	Ch16	Sum	Ch10
1	1,0	2,1	20,3	0,8	0,4	2,4	0,7	0,4	0,5	1,9	0,7	13,2	0,1	0,0	22,7	67,2	46,8
2	0,4	0,4	10,7	0,2	0,1	0,1	0,1	0,1	0,1	0,5	0,2	6,7	0,0	0,0	11,2	30,9	36,0
3	0,5	0,8	1,0	0,4	0,1	0,1	0,1	0,2	0,2	1,9	0,3	0,3	0,0	0,0	1,4	7,5	31,9
4	0,6	1,4	6,9	0,4	0,3	0,2	0,3	0,2	0,2	1,1	0,5	7,5	0,0	0,0	7,8	27,6	44,1
5	0,7	1,5	13,0	0,6	0,4	0,2	0,5	0,3	0,3	0,8	1,1	12,2	0,1	0,0	13,8	45,5	24,7
6	1,1	1,7	13,0	1,0	0,5	0,4	0,6	0,4	0,4	2,0	1,3	10,7	0,1	0,0	14,7	47,8	45,0
7	0,7	1,2	9,1	0,7	0,3	0,2	0,2	0,3	0,3	1,9	0,4	6,9	0,0	0,0	10,9	33,2	45,6
8	0,8	1,7	9,3	0,6	0,4	0,2	0,3	0,3	0,2	1,0	0,7	7,4	0,1	0,0	11,9	34,8	25,9
9	0,5	1,0	6,7	0,4	0,2	0,1	0,3	0,3	0,3	1,1	0,5	12,1	0,1	0,0	9,1	32,7	47,7
10	0,7	1,3	4,0	0,5	0,3	0,1	0,3	0,3	0,3	1,2	0,6	9,2	0,1	0,0	6,2	25,0	41,9
11	0,6	1,0	11,4	0,5	0,3	0,1	0,3	0,2	0,2	1,2	1,3	8,1	0,1	0,0	13,3	38,8	39,4
12	0,6	1,0	7,9	0,4	0,2	0,2	0,4	0,3	0,1	0,8	1,1	5,1	0,0	0,0	11,2	29,4	40,9
13	0,5	1,0	3,3	0,7	0,4	0,3	0,3	0,3	0,2	1,7	0,5	8,8	0,1	0,0	7,1	25,3	27,2
14	0,6	1,0	4,5	0,6	0,2	0,2	0,4	0,3	0,2	1,2	0,6	8,1	0,0	0,0	8,3	26,3	45,0
15	0,5	1,0	4,7	0,5	0,3	0,2	0,2	0,2	0,2	1,0	0,5	8,8	0,0	0,0	9,4	27,5	42,2
16	0,6	1,5	2,7	0,5	0,3	0,1	0,3	0,3	0,3	1,0	0,3	8,6	0,1	0,0	6,6	23,0	47,9
17	0,8	1,6	6,7	0,6	0,3	0,3	0,3	0,4	0,3	1,7	0,6	9,6	0,1	0,0	7,8	31,0	38,9
18	0,5	1,0	2,3	0,5	0,2	0,1	0,1	0,1	0,2	0,7	0,4	9,6	0,0	0,0	4,6	20,5	30,4
19	0,8	1,6	6,8	0,9	0,3	0,2	0,3	0,2	0,4	1,0	0,8	10,6	0,1	0,0	6,7	30,8	30,7
20	1,0	1,6	5,8	0,7	0,2	0,2	0,4	0,3	0,3	1,4	0,6	9,8	0,0	0,0	11,0	33,6	37,2
21	0,6	1,0	1,4	0,6	0,2	0,1	0,2	0,3	0,2	1,3	0,5	5,4	0,1	0,0	0,0	12,0	51,0

(b) Test2

Figure 8.3.: Average channel readings throughout the whole duration of the tests.

8.8. Channel assessment algorithms

In section 8.7 it was reasoned that the channel footprint tests in section 7.9 had limited quality because of various error sources. Still the different measurements does give an impression of how the different algorithms behave compared to each other, even if the test conditions are not as realistic as they should have been, enabling some discussion regarding the different options.

8.8.1. Envelope

In test 1, figure 7.9, a pure envelope is used to capture the detected signals. This solution seems to present a viable ranking of the channels, with both tests resulting in at least one fully green channel from the AP's point of view, the rest of the channels are all colored in fairly different shades, with the channels where the nodes register more or less continuous noise colored with the darker shades of orange. This does give the AP a very good foundation for choosing a new channel should the need arise, and as such the envelope might not be a bad choice.

8.8.2. Average

Test 2, 3 and 4 in figure 7.9 uses the average over 5, 10 and 20 measurements. The first obvious thing to note here is that the channel footprint generally is very clean and a lot of channels are listed to be completely clean throughout the whole test. This is as expected, as was noted in section 3.8.5, since the average value will spread any occasional noise over a number of samples, making the numerical value significantly lower unless it registers a continuous stream of high readings. Thus this approach is likely to require lower limits than the others in order to provide the same channel ranking.

Generally it is not desired to use an approach that gives as many completely green channels as is the case with the average tests, but the second test run of test two shows that it could work very well under the right conditions. It seems likely that with some tuning of the limits and possibly also the averaging period this solution could work very well.

8.8.3. Envelope with filter

The tests using a factor on the envelope have varying results. Firstly it seems clear that the smoothing factor only makes things worse. The tests where this factor was not zero are test 9 to 12 in figure 7.11. The footprints gathered here tends to give a high number of channels that are estimated to be very noisy, making them hard to rank effectively as they all appear to be very bad choices. Thus it can be reasoned that if the envelope with filtering is to be used the smoothing factor can be removed altogether.

Test 5 to 8 in figure 7.10 have reasonable results overall, but it is clear that a maximum factor of 50 is too low when there is a lot of noise, the footprints from the first test run does rank the channels fairly well, but in test run two it is quite clear that the ranking gets better when the limit is increased to 100.

Test 13 and 14 in figure 7.12 has the maximum factor increased to 150 and lastly 200. This seems to give even more completely green channels during test run one, but in test run two these two tests seems to report even worse channel estimates to the AP than test 7 and 8. This makes no sense however and it has to be assumed that the cause of this discrepancy is an increase in outside noise rather than the actual behavior of the algorithm. Once again the poor quality of the tests comes to view and it is clear that more tests should be conducted before any final decisions are made.

8.8.4. Conclusion

After this quick analysis the only certain conclusion is that the smoothing factor has little to no positive effect and can be skipped. Other than that the choice of formula is still an open question and all options do appear to present viable solutions.

There is no question that the simplest choice is the pure envelope, as this requires only a single value to be stored for each channel, and also requires hardly any calculations when storing the measurements.

Averaging will clearly need other limits than what was used in these tests, but test two in the second test run clearly shows that the channel ranking for the AP can become very good with this approach when the conditions are right.

All in all, the main thing to take from the tests is that the configuration of the algorithms should preferably be changed based on the total level of noise in order for the assessment to be the most effective.

8.8.5. Channel ranking on nodes

The suggested dynamic configuration rise the question if another approach should be used instead of what is the case now. Instead of having the nodes report the energy readings directly to the AP, decoded to a compact format, the nodes could perform the ranking locally. Then the channels could be categorizing into groups indicating what channels the node would prefer that the AP moved to if a channel swap was to be made.

This way the channels that are the noisiest from the node's point of view will be in group three regardless of their actual numerical energy level. The channels with the lowest noise level will be in group zero. This will rank the channels compared to each other instead of the way it effectively works now, where the channels are ranked individually compared to a static noise limit.

It might still be sensible to keep the static noise limits in mind when populating the groups. One possibility is to define a minimum and maximum size for the groups, where the noise limits can be used to determine what group to put the channel in if the minimum size limits of the groups are already met.

One can imagine that each group should have a minimum of two and a maximum of six channels in it. The two channels with the lowest energy reading will automatically be placed in group zero no matter what the actual energy reading is. Likewise for the two channels with the highest readings, that will be placed in group two. After this the third lowest channel could be placed in group zero if it has energy below X, or in group one if it has energy above X, this goes on with the next channel in line until group one has two channels in it, at which point the next channel is placed in group one with energy below Y or in group two with energy above Y.

Group three is still limited to active channels only, as it is vital that the AP's has a proper knowledge about this at all times. Also it is obvious that the minimum limits for the groups have to be surpassed if a sufficiently high number of channels belong in the active group.

With the transition to this approach the need for dynamic configuration of the algorithms should not be needed, as the ranking will be self regulating and to some extent independent of the actual noise levels. Still the nodes has to base the ranking on energy measurements, meaning that the question of what approach to use still remains. This should be further evaluated at a later stage after running a new set of tests where the major drawbacks of the currently executed ones are improved and corrected.

8.9. Protocol parameters

During the implementation process limited effort has been put into fine tuning of the protocol parameters, other than the ones that are critical to the functionality. Also the configuration of hardware functions like the CCA has been largely ignored, with the default settings being used in most cases. This means that there might be room for improvement by adjusting particular settings to be better suited for this particular scenario.

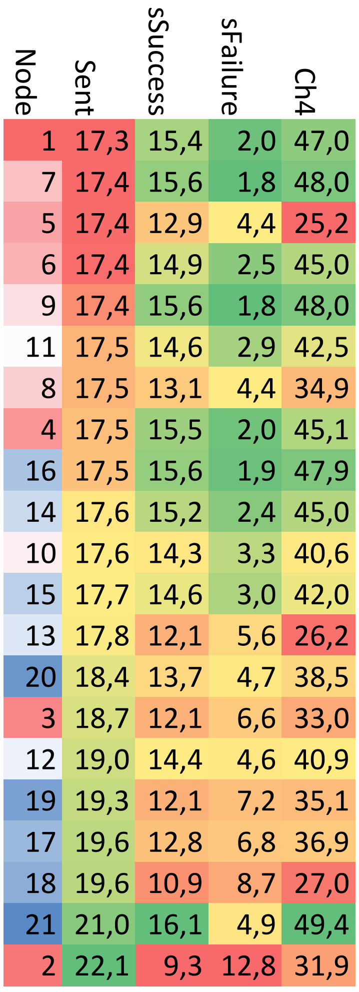
8.9.1. CSMA/CA behavior

In figure 8.4 the average traffic measurements for the nodes during test one in section 7.9 are shown, sorted by the number of initiated transmissions. As seen there is a fairly large spread in the number of transmissions among the nodes, but it is also clear that the nodes getting the highest numbers are getting a significant amount of false positives, causing a very high amount of lost packets due to collisions.

The exception is node 21 that has the overall highest number of successful transmissions, but this is expected given that it is located around one third of the distance from the AP compared to the other nodes.

More than half the nodes report a number of accesses in the range of 17.3 to 17.8, then follows the rest of the nodes with increasingly higher numbers up to a maximum of more than 22. The reported number of failures follows more or less the same sorting as the number of accesses. However there is an obvious link between the number of failures and the signal strength the node registers on the active channel. This indicates that part of the reason for the transmission failures is likely to be the transmission characteristics of the antennas, or rather, the lack of proper antennas.

In section 5.2.1 the collision window was described and estimated, here it was said that the CCA of one node can overlap slightly with the transmission of another while still returning a success, since the result is the average over eight symbols. This possible overlap will become bigger and bigger as the detected signal decreases, meaning that nodes with poor reception characteristics will end up with a bigger collision window and subsequently a higher number of collisions. This exact same effect will also come



Node	Sent	sSuccess	sFailure	Ch4
1	17,3	15,4	2,0	47,0
7	17,4	15,6	1,8	48,0
5	17,4	12,9	4,4	25,2
6	17,4	14,9	2,5	45,0
9	17,4	15,6	1,8	48,0
11	17,5	14,6	2,9	42,5
8	17,5	13,1	4,4	34,9
4	17,5	15,5	2,0	45,1
16	17,5	15,6	1,9	47,9
14	17,6	15,2	2,4	45,0
10	17,6	14,3	3,3	40,6
15	17,7	14,6	3,0	42,0
13	17,8	12,1	5,6	26,2
20	18,4	13,7	4,7	38,5
3	18,7	12,1	6,6	33,0
12	19,0	14,4	4,6	40,9
19	19,3	12,1	7,2	35,1
17	19,6	12,8	6,8	36,9
18	19,6	10,9	8,7	27,0
21	21,0	16,1	4,9	49,4
2	22,1	9,3	12,8	31,9

Figure 8.4.: Average traffic throughout TestresultsChMgmt21Nodes15x200Sec-20-05-13.

into play when the node in question gains genuine access to the channel. Its poor transmission characteristics will make the overlap portion of the collision window of other nodes larger, making them more likely to interrupt the transmission. Two nodes with poor transmission and reception characteristics will thus have an even larger collision window between them. This effect does to some extent explain the behavior seen in figure 8.4. In order to further investigate this, new tests should be conducted using the exact same setup, but with proper antennas mounted on all nodes to see if that evens out the statistics between them.

In addition to the proper assembly of the nodes different hardware settings can be attempted. The CCA can be configured to trigger on decodable signals, any energy above a given threshold, or a combination of the two. Here the default settings have been used for all parameters, with the default being the combination. This means that there is a possibility that other settings can be better suited for this particular usage, in particular the energy thresholds can be investigated further.

Random delay

Currently the protocol use the same maximum value for the random delay in the CSMA/CA algorithm each time, no matter how many times the node has already attempted to access the channel. In a regular CSMA/CA procedure this limit will increase each time, making the sender back off for longer periods as the number of failed accesses increase. The benefit of increasing the limit is that the nodes will automatically access the channel more seldom when the traffic is at a peak. This will decrease the risk for collisions as it will decrease the number of simultaneous CCA's on average.

Because of this it could be a good idea to change the behavior of the protocol to use increasing limits, as it is clear from the measured results that the number of packets lost due to collisions is significant. It is possible to perform tests using a higher random delay limit first, as that is a very simple modification to do. Based on the results in these tests it is then possible to further evaluate the need for progressively increasing limits. Generally it is reasonable to believe that this change will be well worth the effort and should be attempted.

8.9.2. Beacon size

In order to improve the traffic capacity from the server to the nodes the one major parameter is the beacon size. This parameter has been estimated fairly roughly during implementation, and can as such be maximized further. In order to achieve the best performance here the two things to improve are the processing time required by the nodes after the last beacon byte has been transmitted, and the portion of the subperiod needed for the timing buffer.

The link between the timing buffer and the beacon size is very simple, as the beacon size can be incremented by one byte for each two symbols the timing buffer is decremented.

The processing time however can either be measured through analysis of the assembly code, or through trial and error. The simplest way to do this would be to gradually increment the size limit by one and perform a stress test for each step. When the beacon gets too large the result will be that the nodes are unable to complete the processing before the next beacon trigger, making then unable to transmit the acknowledge within the acknowledge window. This will lead to the AP experiencing failed transmissions. This way the tipping point can be found and the limit can be set accordingly. It would still be recommended to include a small buffer of at least 2-4 symbols to take into account that the protocol might be compiled using different optimization flags at a later stage, causing the processing time to change.

Message limit Another aspect of the beacons is the number of messages available. As of now the limit is set to three, but the beacon header does have one more bit available, meaning that, unless this bit is needed for something else at a later stage, the message counter can be expanded to three bits.

If this is done the limiting factor would be the number of acknowledgements that can fit in the acknowledge window rather than the counter. Given the current configuration the acknowledge window is 126 symbols as stated in section 3.9.2. In section 2.5.5 the size of a single acknowledgement is set to 25 symbols, this includes the 20 "on-air" symbols of the frame plus 5 symbols to make sure there is no overlapping even if the nodes are not perfectly synchronized. This would mean that it should be possible to fit 5 acknowledge frames into the full acknowledge window.

How many messages the AP's are actually able to insert into the beacon would still depend directly on the size of the messages and the number of outgoing acknowledgements, but by increasing the limit to five we at least make sure that the protocol does not limit the traffic more than the physical limits in any case.

9. Conclusion

Throughout the work with this thesis a custom specialized protocol has been designed, implemented and tested with actual hardware. The protocol layout has been defined with specialized frames, timings and communication sequences.

The main abilities of the protocol are the following:

- One to many communication through any number of access points.
- Outgoing messages packed into beacons transmitted regularly by the access points.
- Incoming messages transmitted using CSMA/CA within the access window on any active channel.
- Nodes continuously scanning all channels, enabling communication with all access points within range.
- Nodes automatically detecting when one, or all, access points goes out of range or offline.
- Access points able to seamlessly swap channels when needed.

Protocol core The resulting implementation serves as a usable proof of concept for the protocol layout. The protocol core with transmissions on a single channel has been verified to work perfectly, giving single channel results that are very close to the expected theoretical values.

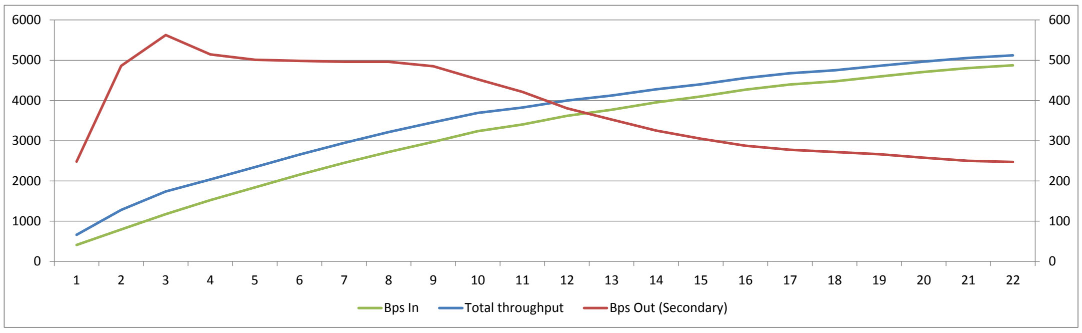
There has been identified multiple issues both regarding the test setup and the protocol that is likely to penalize the throughput, also the parameters of the protocol has seen limited tuning and all in all this points towards the throughput possibly being increased further if the development is continued.

In figure 9.1 the average measured traffic statistics with one channel and different numbers of nodes are presented, the test in question is described in section 7.10.

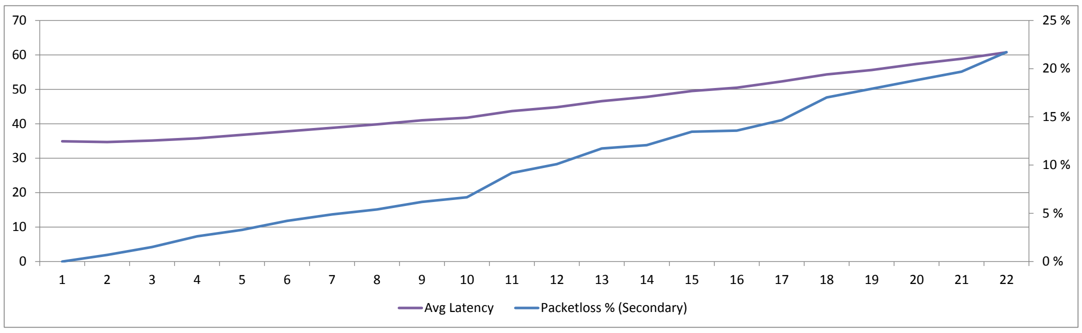
Channel dynamism The procedure of swapping channels has proven to work perfectly, and the reported energy footprints from the nodes allows the AP to avoid Wi-Fi channels and other compeeting radio signals.

The automatic assessment of channel quality however, is not fully operational in its current state and will need significant improvement before it can be used in practice. A number of possible extensions and adjustments has been proposed, and is likely to remove the current malfunctions. This will enable it to better cover all possible scenarios.

Still the protocol is perfectly usable even without this feature, as it has no effect on the system in a stable and static environment. Also it is possible for the application to initiate a channel swap manually if it detects a drop in throughput or an increase in the number of dropped packets. The AP is currently able to avoid the most noisy channels when such a swap is initiated by using the energy reports from the nodes.



(a) Throughput in bytes per second.



(b) Latency and packet loss percentage.

Figure 9.1.: Resulting traffic parameters with 1 to 22 nodes. 14 byte messages from nodes, 8 byte messages from AP's. Each node is atempting to transmit messages continously.

Access point synchronization In order to enable the use of multiple AP's a global synchronization is essential. This functionality has proven to require more advanced procedures than what was initially developed and as such this is not currently at the level of accuracy required for the system to work flawlessly.

Some obvious flaws has come to light after analyzing the tests, and other adjustments and changes has been discussed that is likely to result in a significantly better solution.

The currently observed behavior of the system, with the existing flaws and malfunctions, combined with the proposed changes, makes it reasonable to assume that a working solution is well within reach given some more development, tuning, and new rounds of testing.

Final conclusion The results gained through the work with the thesis is considered a success, and the fundamental concept of the protocol has been proven to work in a practical case. The aspects of the system that is not in working condition have been analyzed and discussed, and the errors most likely causing the experienced issues have been located and corrected theoretically. Given some more work with implementation, debugging and testing it is considered very likely that the protocol will provide a very good solution for the target application.

10. Future work

With the current state of the system there is a significant list of possible future work. Some parts are needed to get the system fully operational as intended, others are non-critical but will significantly increase the flexibility and stability of the protocol once in place. Lastly the overall performance can be improved and adjusted to push the system to its limits both in terms of throughput, and in terms of processing and memory usage on the microcontroller. Also there are several other services that the protocol could provide that have not been implemented by now since it does not influence the normal operation of the system. This includes a specialized broadcast mode and functionality for transmitting larger portions of data from a single, or a small number of nodes to the server. Below the different tasks are summarized and ranked, with features that are needed to put the system in use are considered most critical.

1. Complete the synchronization functionality.
   - Remake the synchronization function.
   - Consider increasing the synchronization frequency.
   - Better match the beacon frequency and the sync frequency.
   - Test and verify synchronization with multiple access points.
2. Implement the remaining portions of the interface.
   - Implement the network ID blacklist and the "don't care" network ID.
   - Adjust buffers to use byte array pointers instead of static arrays.
3. Fully define and implement broadcast mode.
   - Enable broadcast of larger files from the server to all nodes.
   - Consider the possibility of a dedicated data transmission mode for the nodes to transmit larger files to the server when needed.
4. Improve the automatic channel assessment.
   - Register the throughput to better determine if a corrupted frame is truly caused by noise and outside interference.
   - Add detection of missing/erroneous/corrupted acknowledgements to detect competing access points.
   - Test channel swap functionality and channel energy footprint in a realistic environment and with moving nodes.
5. Evaluate the effect of increased backoff periods for the CSMA/CA algorithm.

  - Test the system with higher, static, limits on the random backoff period.
  - Implement progressively increasing limits and test the performance.

6. Change channel reporting approach.
   - Have the nodes rank the channels based on energy readings and sorted in groups before relaying information to AP.
   - AP tracks incoming data with a slow envelope.
   - Grouping based partly on group size limits and partly on fixed energy limits.
7. Further investigate the node deadlock issue to locate the actual reason for the error.
   - Undo the interrupt routine swap that removed the issue to verify that it is still present.
   - Review all buffers and pointer to assure correct usage.
   - Improve test application to supply more and better debug information.
8. Improve and test timings and parameters to enhance capacity and decrease overhead.
   - Measure computational overhead on the nodes when receiving a beacon and increase maximum beacon size accordingly.
   - Measure maximum synchronization offset and adjust timing buffer and maximum beacon size accordingly.
   - Evaluate the possibility of increasing the maximum number of messages per beacon, depending largely on the maximum beacon size and the maximum number of acknowledgements possible within the acknowledge phase.
   - Evaluate the channel assessment limits on the nodes to best suit a realistic environment.
   - Consider the possibility of using an increased bit rate supported by the hardware to further enhance the capacity of the protocol.
9. Rewrite protocol procedures to minimize memory usage and computational overhead.

Abbreviations

802.15.4	IEEE standard for low rate PAN's and sensor networks.
Access Window	The portion of the beacon period where anyone can transmit data frames using CSMA/CA.
AP	Access Point
Beacon	Specialized frame transmitted regularly by the access point.
Beacon Period	Protocol definition giving the time between two beacons on the same channel, consists of 16 subperiods.
CCA	Clear Channel Assesment
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
ED	Energy Detection
FCS	Frame Check Sequence
GTS	Guaranteed Time Slot
MAC	Media Access Control. The second lowest level of the OSI network model.
PAN	Personal Area Network
PHR	PHY header
PHY	PHYsical layer. Lowest level of the OSI network model.
POE	Power Over Ethernet
PSDU	Physical Layer Service Data Unit
RSSI	Received Signal Strength Indication
SFD	Start of Frame Delimiter
SoC	System On Chip
Subperiod	Protocol definition giving the fundamental timing unit.
UART	Universal Asynchronous Received/Transmitter

Bibliography

[1] Ragnar Stuhaug,
Evaluation of protocols for Real-time radio systems running on 2.4GHz,
NTNU, 2012.

[2] AutoStore System Official website,
http://www.autostoresystem.com/, 2012.

[3] ATMEL Corporation,
ATmega128RFA1 datasheet - 8266D-MCU Wireless-06/12, 06-2012.

[4] Dresden Elektronik - *www.dresden-elektronik.de*,
User Manual Radio Modules - deRFmega128-22C02,
Document Version V1.4 2011-08-19.

[5] Institute of Electrical and Electronics Engineers, Inc.,
IEEE Std. 802.15.4-2006,
IEEE Standard for Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). New York: IEEE Press. 2006.

[6] HP InfoTech - CodeVisionAVR,
http://hpinfotech.ro/index.html,
Version: 2.60 Standard, 2012

[7] Atmel Corporation - AtmelStudio6,
http://www.atmel.com/microsite/atmel_studio6/,
Version: 6.1.2562, 2013

List of Figures

List of Tables

Appendices

A. Theoretical results

The full spreadsheet for the theoretical calculations can be seen in the supplied archive file. In table A.1, A.2, A.3, and A.4 the results calculated for the four different test scenarios are presented.

The parameters for the models are as given in chapter 5 with the exception of message sizes and access buffer that vary from test to test.

Note that the values presented here are the calculations for a full second, meaning that the formulas in chapter 5 are multiplied by the beacon frequency.

Note also that due to the error in the test application on the nodes they report values from the previous 1.04 second instead of the last one second. This means that the theoretical values must be increased by 4% to be directly comparable to the node reports. This is done in the afflicted plots throughout chapter 7.

	Node sum per sec				Per node per sec						AP per sec		
Nodes	Transmissions	Success	Failed	CSMA Miss	Transmissions	Success	Failure	CSMAMiss	Received-E1	Received-E2	Transmitted-E1	Transmitted-E2	Received
1	31	31	0,0	0 %	31,0	31,0	0,0	0,0	31,0	31,0	31,0	31,0	31
2	62	62	0,1	3 %	31,0	30,9	0,1	1,0	31,0	31,0	62,0	62,0	62
3	93	93	0,4	6 %	31,0	30,9	0,1	2,1	29,1	22,3	87,4	67,0	93
4	124	123	0,8	10 %	31,0	30,8	0,2	3,3	20,5	15,7	81,9	62,7	123
5	155	153	2,0	13 %	31,0	30,6	0,4	4,5	15,3	12,4	76,5	61,9	153
6	186	183	2,8	16 %	31,0	30,5	0,5	5,8	11,8	10,0	71,0	60,3	183
7	217	213	3,8	19 %	31,0	30,5	0,5	7,3	9,4	7,7	65,5	53,7	213
8	248	242	6,1	22 %	31,0	30,2	0,8	8,9	7,5	5,3	60,3	42,8	242
9	275	267	7,4	25 %	30,5	29,7	0,8	10,2	6,2	3,9	55,7	35,0	267
10	296	285	10,5	27 %	29,6	28,5	1,0	11,0	5,2	3,2	52,4	32,4	285
11	317	305	12,2	29 %	28,8	27,7	1,1	12,0	4,4	2,8	48,8	31,3	305
12	338	322	16,0	31 %	28,2	26,9	1,3	13,0	3,8	2,6	45,6	30,9	322
13	357	339	18,2	33 %	27,5	26,1	1,4	13,8	3,3	2,3	42,6	30,4	339
14	376	353	22,8	35 %	26,9	25,2	1,6	14,7	2,9	2,1	40,0	29,3	353
15	394	366	27,9	37 %	26,3	24,4	1,9	15,5	2,5	1,8	37,8	27,4	366
16	411	380	30,7	39 %	25,7	23,7	1,9	16,3	2,2	1,5	35,2	23,9	380
17	427	390	36,6	41 %	25,1	23,0	2,2	17,1	2,0	1,2	33,3	20,5	390
18	442	399	43,0	42 %	24,6	22,2	2,4	17,8	1,8	0,9	31,7	17,1	399
19	456	410	46,3	44 %	24,0	21,6	2,4	18,6	1,6	0,7	29,7	12,7	410
20	470	417	53,5	45 %	23,5	20,8	2,7	19,2	1,4	0,5	28,5	10,2	417
21	483	422	61,1	46 %	23,0	20,1	2,9	19,8	1,3	0,4	27,5	8,3	422
22	496	431	65,1	48 %	22,5	19,6	3,0	20,5	1,2	0,3	26,0	5,8	431
23	507	434	73,5	49 %	22,1	18,9	3,2	21,0	1,1	0,2	25,4	5,0	434
24	518	436	82,4	50 %	21,6	18,2	3,4	21,5	1,0	0,2	25,0	4,4	436
25	529	437	91,7	51 %	21,2	17,5	3,7	22,0	1,0	0,2	24,8	4,2	437
26	539	437	102	52 %	20,7	16,8	3,9	22,5	1,0	0,2	24,8	4,2	437
27	548	442	107	53 %	20,3	16,4	3,9	22,9	0,9	0,1	24,0	3,3	442
28	557	440	117	54 %	19,9	15,7	4,2	23,3	0,9	0,1	24,2	3,6	440
29	566	438	128	55 %	19,5	15,1	4,4	23,6	0,9	0,1	24,7	4,1	438
30	574	434	140	56 %	19,1	14,5	4,7	24,0	0,8	0,2	25,3	4,9	434
31	582	436	146	56 %	18,8	14,1	4,7	24,3	0,8	0,1	25,0	4,4	436
32	589	431	158	57 %	18,4	13,5	4,9	24,5	0,8	0,2	25,9	5,6	431
33	596	426	170	58 %	18,1	12,9	5,2	24,8	0,8	0,2	26,9	7,2	426
34	603	419	184	59 %	17,7	12,3	5,4	25,0	0,8	0,3	28,1	9,4	419
35	609	412	197	59 %	17,4	11,8	5,6	25,2	0,8	0,3	29,4	12,0	412
36	615	404	211	60 %	17,1	11,2	5,9	25,4	0,9	0,4	30,8	15,1	404
37	621	403	218	60 %	16,8	10,9	5,9	25,6	0,8	0,4	31,1	15,7	403
38	626	393	233	61 %	16,5	10,3	6,1	25,7	0,9	0,5	32,8	19,3	393
39	631	383	248	61 %	16,2	9,8	6,4	25,9	0,9	0,6	34,6	22,8	383
40	636	373	264	62 %	15,9	9,3	6,6	26,0	0,9	0,6	36,5	25,9	373
41	641	362	280	62 %	15,6	8,8	6,8	26,1	0,9	0,7	38,5	28,2	362
42	646	358	288	63 %	15,4	8,5	6,9	26,2	0,9	0,7	39,2	28,7	358
43	650	346	304	63 %	15,1	8,0	7,1	26,2	1,0	0,7	41,4	30,0	346
44	655	333	322	64 %	14,9	7,6	7,3	26,3	1,0	0,7	43,7	30,6	333
45	659	320	339	64 %	14,6	7,1	7,5	26,3	1,0	0,7	46,2	31,0	320
46	663	306	357	65 %	14,4	6,6	7,8	26,4	1,1	0,7	48,7	31,3	306
47	666	291	375	65 %	14,2	6,2	8,0	26,4	1,1	0,7	51,4	31,9	291
48	670	285	385	65 %	14,0	5,9	8,0	26,4	1,1	0,7	52,4	32,4	285
49	673	270	404	66 %	13,7	5,5	8,2	26,4	1,1	0,7	55,2	34,5	270
50	677	254	423	66 %	13,5	5,1	8,5	26,4	1,2	0,8	58,1	38,6	254
51	680	237	443	66 %	13,3	4,7	8,7	26,4	1,2	0,9	61,2	44,6	237
52	683	220	463	67 %	13,1	4,2	8,9	26,4	1,2	1,0	64,3	51,3	220
53	686	202	484	67 %	12,9	3,8	9,1	26,4	1,3	1,1	67,5	56,9	202
54	689	195	494	67 %	12,8	3,6	9,2	26,4	1,3	1,1	68,8	58,5	195
55	692	176	515	68 %	12,6	3,2	9,4	26,3	1,3	1,1	72,2	60,9	176
56	695	157	537	68 %	12,4	2,8	9,6	26,3	1,4	1,1	75,6	61,8	157
57	697	138	559	68 %	12,2	2,4	9,8	26,3	1,4	1,1	79,2	62,2	138
58	700	118	582	68 %	12,1	2,0	10,0	26,2	1,4	1,1	82,8	63,1	118
59	702	98	604	69 %	11,9	1,7	10,2	26,2	1,5	1,1	86,5	65,9	98
60	705	89	616	69 %	11,7	1,5	10,3	26,1	1,5	1,1	88,2	68,1	89

Table A.1.: Theoretical results with 8 byte messages in both directions and correct access buffer.

	Node sum per sec				Per node per sec						AP per sec		
Nodes	Transmissions	Success	Failed	CSMA Miss	Transmissions	Success	Failure	CSMAMiss	Received-E1	Received-E2	Transmitted-E1	Transmitted-E2	Received
1	31	31	0,0	0 %	31,0	31,0	0,0	0,0	31,0	31,0	31,0	31,0	31
2	62	62	0,2	4 %	31,0	30,9	0,1	1,1	31,0	31,0	62,0	62,0	62
3	93	93	0,5	7 %	31,0	30,8	0,2	2,4	29,2	22,3	87,5	67,0	93
4	124	123	1,0	11 %	31,0	30,8	0,2	3,7	20,5	15,7	81,9	62,8	123
5	155	153	2,4	14 %	31,0	30,5	0,5	5,1	15,3	12,4	76,5	61,9	153
6	186	183	3,3	18 %	31,0	30,4	0,6	6,7	11,8	10,1	71,1	60,3	183
7	217	213	4,5	21 %	31,0	30,4	0,6	8,4	9,4	7,7	65,6	53,9	213
8	242	235	7,1	24 %	30,2	29,3	0,9	9,6	7,7	5,7	61,6	45,6	235
9	262	254	8,7	26 %	29,2	28,2	1,0	10,5	6,5	4,3	58,2	38,7	254
10	283	271	12,3	29 %	28,3	27,1	1,2	11,5	5,5	3,4	55,0	34,3	271
11	303	289	14,4	31 %	27,5	26,2	1,3	12,4	4,7	2,9	51,8	32,1	289
12	321	302	18,9	33 %	26,8	25,2	1,6	13,3	4,1	2,6	49,3	31,4	302
13	340	316	24,0	35 %	26,1	24,3	1,8	14,2	3,6	2,4	46,9	31,0	316
14	356	329	26,8	37 %	25,4	23,5	1,9	15,0	3,2	2,2	44,4	30,8	329
15	372	340	32,9	39 %	24,8	22,6	2,2	15,9	2,8	2,0	42,5	30,4	340
16	387	347	39,5	41 %	24,2	21,7	2,5	16,6	2,6	1,9	41,1	29,9	347
17	402	359	43,0	42 %	23,6	21,1	2,5	17,4	2,3	1,7	39,1	28,6	359
18	415	364	50,5	44 %	23,0	20,2	2,8	18,0	2,1	1,5	38,0	27,7	364
19	428	369	58,6	45 %	22,5	19,4	3,1	18,7	2,0	1,4	37,2	26,7	369
20	440	377	62,9	47 %	22,0	18,8	3,1	19,3	1,8	1,2	35,7	24,8	377
21	451	379	71,8	48 %	21,5	18,1	3,4	19,8	1,7	1,1	35,3	24,1	379
22	462	381	81,4	49 %	21,0	17,3	3,7	20,4	1,6	1,1	35,0	23,6	381
23	472	381	91,4	50 %	20,5	16,6	4,0	20,9	1,5	1,0	35,0	23,6	381
24	482	385	96,7	52 %	20,1	16,1	4,0	21,3	1,4	0,9	34,2	22,2	385
25	491	384	107,6	53 %	19,6	15,3	4,3	21,8	1,4	0,9	34,5	22,8	384
26	500	381	119	54 %	19,2	14,6	4,6	22,2	1,3	0,9	35,0	23,7	381
27	508	377	131	54 %	18,8	14,0	4,9	22,5	1,3	0,9	35,8	24,8	377
28	516	372	144	55 %	18,4	13,3	5,1	22,9	1,3	0,9	36,6	26,1	372
29	523	373	150	56 %	18,0	12,9	5,2	23,2	1,3	0,9	36,4	25,8	373
30	530	367	164	57 %	17,7	12,2	5,5	23,5	1,3	0,9	37,6	27,3	367
31	537	359	178	58 %	17,3	11,6	5,7	23,7	1,3	0,9	38,9	28,5	359
32	543	351	192	59 %	17,0	11,0	6,0	23,9	1,3	0,9	40,4	29,5	351
33	549	342	207	59 %	16,6	10,4	6,3	24,2	1,3	0,9	42,0	30,2	342
34	555	340	215	60 %	16,3	10,0	6,3	24,4	1,2	0,9	42,4	30,3	340
35	561	330	230	60 %	16,0	9,4	6,6	24,5	1,3	0,9	44,2	30,7	330
36	566	319	247	61 %	15,7	8,9	6,9	24,7	1,3	0,9	46,3	31,0	319
37	571	307	264	62 %	15,4	8,3	7,1	24,8	1,3	0,8	48,4	31,2	307
38	576	295	281	62 %	15,1	7,8	7,4	24,9	1,3	0,8	50,7	31,7	295
39	580	290	290	63 %	14,9	7,4	7,4	25,0	1,3	0,8	51,5	32,0	290
40	584	277	308	63 %	14,6	6,9	7,7	25,1	1,3	0,8	54,0	33,4	277
41	589	262	326	64 %	14,4	6,4	8,0	25,2	1,4	0,9	56,6	36,2	262
42	593	247	345	64 %	14,1	5,9	8,2	25,3	1,4	1,0	59,3	40,8	247
43	596	232	365	65 %	13,9	5,4	8,5	25,3	1,4	1,1	62,2	46,9	232
44	600	215	385	65 %	13,6	4,9	8,7	25,3	1,5	1,2	65,1	53,1	215
45	604	209	395	65 %	13,4	4,6	8,8	25,4	1,5	1,2	66,4	55,2	209
46	607	191	416	66 %	13,2	4,2	9,0	25,4	1,5	1,3	69,5	59,2	191
47	610	173	437	66 %	13,0	3,7	9,3	25,4	1,5	1,3	72,8	61,1	173
48	614	155	459	67 %	12,8	3,2	9,6	25,4	1,6	1,3	76,1	61,9	155
49	617	136	481	67 %	12,6	2,8	9,8	25,4	1,6	1,3	79,6	62,2	136
50	619	116	504	67 %	12,4	2,3	10,1	25,4	1,7	1,3	83,2	63,3	116
51	622	107	515	68 %	12,2	2,1	10,1	25,4	1,7	1,3	84,8	64,2	107
52	625	87	538	68 %	12,0	1,7	10,4	25,4	1,7	1,3	88,5	68,6	87
53	628	65	562	68 %	11,8	1,2	10,6	25,3	1,7	1,4	92,4	76,2	65
54	630	44	586	68 %	11,7	0,8	10,9	25,3	1,7	1,6	93,0	84,4	44
55	633	21	611	69 %	11,5	0,4	11,1	25,3	1,7	1,6	93,0	90,0	21
56	635	-1	636	69 %	11,3	0,0	11,4	25,2	1,7	1,6	93,0	92,4	-1
57	637	-12	649	69 %	11,2	-0,2	11,4	25,2	1,6	1,6	93,0	92,7	-12
58	639	-35	675	69 %	11,0	-0,6	11,6	25,1	1,6	1,6	93,0	93,0	-35
59	642	-59	701	70 %	10,9	-1,0	11,9	25,1	1,6	1,6	93,0	93,0	-59
60	644	-84	728	70 %	10,7	-1,4	12,1	25,0	1,6	1,5	93,0	93,0	-84

Table A.2.: Theoretical results with 11 byte messages from node and 8 byte messages from AP. Access buffer size is according to the incorrect calculation used in the protocol code.

	Node sum per sec				Per node per sec						AP per sec		
Nodes	Transmissions	Success	Failed	CSMA Miss	Transmissions	Success	Failure	CSMAMiss	Received-E1	Received-E2	Transmitted-E1	Transmitted-E2	Received
1	31	31	0,0	0 %	31,0	31,0	0,0	0,0	31,0	31,0	31,0	31,0	31
2	62	62	0,2	4 %	31,0	30,9	0,1	1,3	31,0	31,0	62,0	62,0	62
3	93	93	0,5	8 %	31,0	30,8	0,2	2,6	29,2	22,3	87,5	67,0	93
4	124	123	1,0	12 %	31,0	30,8	0,2	4,1	20,5	15,7	81,9	62,8	123
5	155	153	2,4	16 %	31,0	30,5	0,5	5,7	15,3	12,4	76,5	61,9	153
6	186	183	3,3	20 %	31,0	30,4	0,6	7,5	11,8	10,1	71,1	60,3	183
7	217	211	5,7	23 %	31,0	30,2	0,8	9,5	9,4	7,8	65,9	54,3	211
8	238	231	7,1	26 %	29,8	28,9	0,9	10,5	7,8	5,9	62,3	47,1	231
9	258	248	10,4	29 %	28,7	27,6	1,2	11,5	6,6	4,5	59,2	40,6	248
10	279	267	12,3	31 %	27,9	26,7	1,2	12,7	5,6	3,5	55,8	35,2	267
11	297	280	16,6	33 %	27,0	25,5	1,5	13,6	4,8	3,0	53,3	32,9	280
12	315	296	18,9	36 %	26,2	24,6	1,6	14,6	4,2	2,6	50,5	31,6	296
13	331	307	24,0	38 %	25,5	23,6	1,8	15,5	3,7	2,4	48,5	31,2	307
14	347	317	29,8	40 %	24,8	22,6	2,1	16,4	3,3	2,2	46,6	31,0	317
15	361	329	32,9	42 %	24,1	21,9	2,2	17,2	3,0	2,1	44,5	30,8	329
16	375	336	39,5	43 %	23,5	21,0	2,5	18,0	2,7	1,9	43,2	30,5	336
17	388	342	46,7	45 %	22,8	20,1	2,7	18,7	2,5	1,8	42,2	30,3	342
18	401	350	50,5	47 %	22,3	19,5	2,8	19,4	2,3	1,6	40,6	29,6	350
19	412	354	58,6	48 %	21,7	18,6	3,1	20,1	2,1	1,5	40,0	29,3	354
20	423	356	67,3	49 %	21,2	17,8	3,4	20,7	2,0	1,5	39,6	29,0	356
21	434	357	76,5	51 %	20,6	17,0	3,6	21,3	1,9	1,4	39,4	28,9	357
22	443	362	81,4	52 %	20,1	16,4	3,7	21,7	1,8	1,3	38,5	28,2	362
23	452	361	91,4	53 %	19,7	15,7	4,0	22,2	1,7	1,2	38,7	28,3	361
24	461	359	102,1	54 %	19,2	14,9	4,3	22,7	1,6	1,2	39,1	28,6	359
25	469	355	113,3	55 %	18,7	14,2	4,5	23,1	1,6	1,2	39,7	29,1	355
26	476	351	125	56 %	18,3	13,5	4,8	23,4	1,6	1,1	40,4	29,5	351
27	483	352	131	57 %	17,9	13,0	4,9	23,8	1,5	1,1	40,2	29,4	352
28	490	347	144	58 %	17,5	12,4	5,1	24,1	1,5	1,1	41,3	29,9	347
29	497	340	157	59 %	17,1	11,7	5,4	24,4	1,5	1,0	42,5	30,4	340
30	503	332	171	59 %	16,8	11,1	5,7	24,6	1,5	1,0	43,9	30,7	332
31	509	324	185	60 %	16,4	10,4	6,0	24,8	1,5	1,0	45,4	30,9	324
32	514	322	192	61 %	16,1	10,1	6,0	25,0	1,4	1,0	45,7	30,9	322
33	519	312	207	62 %	15,7	9,5	6,3	25,2	1,4	0,9	47,5	31,1	312
34	524	302	223	62 %	15,4	8,9	6,5	25,3	1,5	0,9	49,4	31,4	302
35	529	290	239	63 %	15,1	8,3	6,8	25,5	1,5	0,9	51,5	32,0	290
36	533	278	255	63 %	14,8	7,7	7,1	25,6	1,5	0,9	53,7	33,2	278
37	538	266	272	64 %	14,5	7,2	7,4	25,7	1,5	1,0	56,0	35,4	266
38	542	261	281	64 %	14,3	6,9	7,4	25,8	1,5	1,0	56,8	36,6	261
39	546	247	299	65 %	14,0	6,3	7,7	25,8	1,5	1,0	59,4	40,9	247
40	549	232	317	65 %	13,7	5,8	7,9	25,9	1,6	1,2	62,0	46,5	232
41	553	217	336	66 %	13,5	5,3	8,2	25,9	1,6	1,3	64,8	52,4	217
42	556	201	355	66 %	13,2	4,8	8,5	26,0	1,6	1,4	67,7	57,1	201
43	560	185	375	67 %	13,0	4,3	8,7	26,0	1,6	1,4	70,7	60,1	185
44	563	178	385	67 %	12,8	4,0	8,7	26,0	1,6	1,4	71,9	60,8	178
45	566	160	405	67 %	12,6	3,6	9,0	26,0	1,7	1,4	75,1	61,7	160
46	569	142	426	68 %	12,4	3,1	9,3	26,0	1,7	1,3	78,4	62,1	142
47	571	124	448	68 %	12,2	2,6	9,5	26,0	1,7	1,3	81,8	62,7	124
48	574	104	470	68 %	12,0	2,2	9,8	26,0	1,8	1,3	85,3	64,7	104
49	577	84	492	69 %	11,8	1,7	10,0	25,9	1,8	1,4	88,9	69,3	84
50	579	64	515	69 %	11,6	1,3	10,3	25,9	1,9	1,5	92,6	76,7	64
51	582	55	527	69 %	11,4	1,1	10,3	25,8	1,8	1,6	93,0	80,3	55
52	584	34	550	70 %	11,2	0,6	10,6	25,8	1,8	1,7	93,0	87,4	34
53	586	12	574	70 %	11,1	0,2	10,8	25,8	1,8	1,7	93,0	91,4	12
54	588	-11	599	70 %	10,9	-0,2	11,1	25,7	1,7	1,7	93,0	92,7	-11
55	590	-33	624	70 %	10,7	-0,6	11,3	25,6	1,7	1,7	93,0	93,0	-33
56	592	-57	649	71 %	10,6	-1,0	11,6	25,6	1,7	1,7	93,0	93,0	-57
57	594	-68	662	71 %	10,4	-1,2	11,6	25,5	1,6	1,6	93,0	93,0	-68
58	596	-92	688	71 %	10,3	-1,6	11,9	25,4	1,6	1,6	93,0	93,0	-92
59	598	-116	714	71 %	10,1	-2,0	12,1	25,4	1,6	1,6	93,0	93,0	-116
60	600	-142	741	72 %	10,0	-2,4	12,4	25,3	1,6	1,5	93,0	93,0	-142

Table A.3.: Theoretical results with 14 byte messages from node and 8 byte messages from AP. Access buffer size is according to the incorrect calculation used in the protocol code.

	Node sum per sec				Per node per sec						AP per sec		
Nodes	Transmissions	Success	Failed	CSMA Miss	Transmissions	Success	Failure	CSMAMiss	Received-E1	Received-E2	Transmitted-E1	Transmitted-E2	Received
1	31	31	0,0	0 %	31,0	31,0	0,0	0,0	31,0	31,0	31,0	31,0	31
2	62	62	0,2	5 %	31,0	30,9	0,1	1,8	31,0	31,0	62,0	62,0	62
3	93	93	0,5	11 %	31,0	30,8	0,2	3,7	29,2	22,3	87,5	67,0	93
4	124	122	1,6	16 %	31,0	30,6	0,4	6,0	20,5	15,7	82,0	62,8	122
5	155	153	2,4	21 %	31,0	30,5	0,5	8,5	15,3	12,4	76,5	61,9	153
6	186	182	4,5	27 %	31,0	30,3	0,7	11,4	11,9	10,1	71,3	60,4	182
7	206	200	5,7	30 %	29,4	28,6	0,8	12,8	9,7	8,2	67,9	57,4	200
8	225	216	8,7	34 %	28,1	27,0	1,1	14,2	8,1	6,6	65,0	52,8	216
9	242	232	10,4	37 %	26,9	25,7	1,2	15,5	6,9	5,2	62,1	46,8	232
10	258	243	14,4	39 %	25,8	24,3	1,4	16,7	6,0	4,2	60,0	42,2	243
11	273	254	18,9	42 %	24,8	23,1	1,7	17,9	5,3	3,5	58,1	38,6	254
12	286	265	21,4	44 %	23,9	22,1	1,8	18,9	4,7	3,0	56,1	35,6	265
13	299	272	26,8	46 %	23,0	20,9	2,1	19,9	4,2	2,6	54,8	34,1	272
14	311	278	32,9	48 %	22,2	19,8	2,3	20,9	3,8	2,4	53,7	33,2	278
15	321	282	39,5	50 %	21,4	18,8	2,6	21,6	3,5	2,2	53,1	32,8	282
16	331	288	43,0	52 %	20,7	18,0	2,7	22,4	3,2	2,0	51,9	32,1	288
17	340	290	50,5	54 %	20,0	17,0	3,0	23,1	3,0	1,9	51,6	32,0	290
18	348	290	58,6	55 %	19,4	16,1	3,3	23,7	2,9	1,8	51,6	32,0	290
19	356	289	67,3	56 %	18,7	15,2	3,5	24,2	2,7	1,7	51,7	32,1	289
20	363	287	76,5	58 %	18,2	14,3	3,8	24,7	2,6	1,6	52,1	32,2	287
21	370	289	81,4	59 %	17,6	13,8	3,9	25,1	2,5	1,5	51,8	32,1	289
22	376	285	91,4	60 %	17,1	13,0	4,2	25,5	2,4	1,5	52,5	32,4	285
23	382	280	102,1	61 %	16,6	12,2	4,4	25,8	2,3	1,4	53,4	32,9	280
24	387	274	113,3	62 %	16,1	11,4	4,7	26,1	2,3	1,4	54,4	33,8	274
25	393	267	125,1	63 %	15,7	10,7	5,0	26,3	2,2	1,4	55,6	35,0	267
26	397	260	137	63 %	15,3	10,0	5,3	26,6	2,2	1,4	57,0	36,8	260
27	402	258	144	64 %	14,9	9,6	5,3	26,7	2,1	1,4	57,4	37,4	258
28	406	249	157	65 %	14,5	8,9	5,6	26,9	2,1	1,4	59,0	40,2	249
29	410	239	171	66 %	14,1	8,2	5,9	27,0	2,1	1,5	60,8	43,8	239
30	413	229	185	66 %	13,8	7,6	6,2	27,1	2,1	1,6	62,7	48,0	229
31	417	217	199	67 %	13,4	7,0	6,4	27,1	2,1	1,7	64,7	52,3	217
32	420	205	215	67 %	13,1	6,4	6,7	27,2	2,1	1,8	66,9	56,1	205
33	423	201	223	68 %	12,8	6,1	6,7	27,2	2,1	1,7	67,8	57,3	201
34	426	188	239	68 %	12,5	5,5	7,0	27,2	2,1	1,8	70,2	59,7	188
35	429	174	255	69 %	12,3	5,0	7,3	27,2	2,1	1,7	72,7	61,1	174
36	432	159	272	69 %	12,0	4,4	7,6	27,2	2,1	1,7	75,3	61,7	159
37	434	144	290	70 %	11,7	3,9	7,8	27,2	2,1	1,7	78,0	62,1	144
38	436	129	308	70 %	11,5	3,4	8,1	27,1	2,1	1,6	80,9	62,5	129
39	439	112	326	71 %	11,2	2,9	8,4	27,1	2,1	1,6	83,8	63,6	112
40	441	105	336	71 %	11,0	2,6	8,4	27,0	2,1	1,6	85,2	64,5	105
41	443	88	355	71 %	10,8	2,1	8,7	27,0	2,2	1,7	88,3	68,3	88
42	445	70	375	72 %	10,6	1,7	8,9	26,9	2,2	1,8	91,5	74,3	70
43	447	52	395	72 %	10,4	1,2	9,2	26,8	2,2	1,9	93,0	81,5	52
44	449	33	416	72 %	10,2	0,7	9,5	26,7	2,1	2,0	93,0	87,7	33
45	450	13	437	73 %	10,0	0,3	9,7	26,6	2,1	2,0	93,0	91,2	13
46	452	-7	459	73 %	9,8	-0,1	10,0	26,5	2,0	2,0	93,0	92,6	-7
47	454	-16	470	73 %	9,7	-0,3	10,0	26,4	2,0	2,0	93,0	92,8	-16
48	455	-37	492	73 %	9,5	-0,8	10,3	26,3	1,9	1,9	93,0	93,0	-37
49	457	-59	515	74 %	9,3	-1,2	10,5	26,2	1,9	1,9	93,0	93,0	-59
50	458	-80	538	74 %	9,2	-1,6	10,8	26,1	1,9	1,9	93,0	93,0	-80
51	459	-103	562	74 %	9,0	-2,0	11,0	25,9	1,8	1,8	93,0	93,0	-103
52	461	-126	586	74 %	8,9	-2,4	11,3	25,8	1,8	1,8	93,0	93,0	-126
53	462	-149	611	75 %	8,7	-2,8	11,5	25,7	1,8	1,8	93,0	93,0	-149
54	463	-161	624	75 %	8,6	-3,0	11,5	25,6	1,7	1,7	93,0	93,0	-161
55	464	-185	649	75 %	8,4	-3,4	11,8	25,5	1,7	1,7	93,0	93,0	-185
56	465	-209	675	75 %	8,3	-3,7	12,1	25,3	1,7	1,7	93,0	93,0	-209
57	467	-235	701	75 %	8,2	-4,1	12,3	25,2	1,6	1,6	93,0	93,0	-235
58	468	-260	728	76 %	8,1	-4,5	12,5	25,1	1,6	1,6	93,0	93,0	-260
59	469	-286	755	76 %	7,9	-4,9	12,8	24,9	1,6	1,6	93,0	93,0	-286
60	470	-313	782	76 %	7,8	-5,2	13,0	24,8	1,6	1,6	93,0	93,0	-313

Table A.4.: Theoretical results with 26 byte messages from node and 8 byte messages from AP. Access buffer size is according to the incorrect calculation used in the protocol code.

B. Archive file contents

Archive/Code
Archive/Code/ASRadioAP
Archive/Code/ASRadioAP/ap_channel_management.c
Archive/Code/ASRadioAP/ap_channel_management.h
Archive/Code/ASRadioAP/ap_radio.c
Archive/Code/ASRadioAP/ap_radio.h
Archive/Code/ASRadioAP/ap_radio_if.h
Archive/Code/ASRadioAP/ASRadioAP.c
Archive/Code/ASRadioAP/data_types.c
Archive/Code/ASRadioAP/data_types.h
Archive/Code/ASRadioAP/mega128RFA1.h
Archive/Code/ASRadioAP/uart.c
Archive/Code/ASRadioAP/uart.h
Archive/Code/ASRadioGeneral
Archive/Code/ASRadioGeneral/channel_management.c
Archive/Code/ASRadioGeneral/channel_management.h
Archive/Code/ASRadioGeneral/general_radio.c
Archive/Code/ASRadioGeneral/general_radio.h
Archive/Code/ASRadioGeneral/general_radio_config.h
Archive/Code/ASRadioGeneral/general_radio_if.h
Archive/Code/ASRadioGeneral/mac_symbol_counter.c
Archive/Code/ASRadioGeneral/mac_symbol_counter.h
Archive/Code/ASRadioGeneral/test_setup.c
Archive/Code/ASRadioGeneral/test_setup.h
Archive/Code/ASRadioNode
Archive/Code/ASRadioNode/ASRadioNode.c
Archive/Code/ASRadioNode/data_types.c
Archive/Code/ASRadioNode/data_types.h
Archive/Code/ASRadioNode/mega128RFA1.h
Archive/Code/ASRadioNode/node_channel_management.c
Archive/Code/ASRadioNode/node_channel_management.h
Archive/Code/ASRadioNode/node_radio.c
Archive/Code/ASRadioNode/node_radio.h
Archive/Code/ASRadioNode/node_radio_if.h
Archive/Code/ASRadioNode/uart.c
Archive/Code/ASRadioNode/uart.h

Table B.1.: Archived codefiles.

Archive/Test results
Archive/Test results/APLogCHMgmtTst21Node15x200sec1.txt
Archive/Test results/APLogCHMgmtTst21Node15x200sec2.txt
Archive/Test results/APLogCHMgmtTst21Node15x200sec3.txt
Archive/Test results/APLogCHMgmtTst21Node15x200sec4.txt
Archive/Test results/APLogChSwap21NodeNoSync1.txt
Archive/Test results/APLogChSwap21NodeNoSync2.txt
Archive/Test results/APLogChSwap21NodeNoSync3APChSwapChange.txt
Archive/Test results/APLogChSwap21NodeNoSync4APChSwapChange.txt
Archive/Test results/APLogChSwap21NodeNoSync5APChSwapChange.txt
Archive/Test results/APLogChSwap21NodeNoSync6APChSwapChange.txt
Archive/Test results/APLogNodeAddition20Node10Sec1.txt
Archive/Test results/APLogNodeAddition20Node10Sec2.txt
Archive/Test results/APLogNodeAddition20Node10Sec3.txt
Archive/Test results/APLogNodeAddition20Node10Sec4NoSync.txt
Archive/Test results/APLogNodeAddition21Node10Sec1AbsNoChSwapNoSync.txt
Archive/Test results/APLogNodeAddition21Node10Sec1NoSync.txt
Archive/Test results/APLogNodeAddition21Node10Sec2AbsNoChSwapNoSync.txt
Archive/Test results/APLogNodeAddition21Node10Sec2NoSync.txt
Archive/Test results/APLogNodeAddition21Node10Sec3AbsNoChSwapNoSync.txt
Archive/Test results/APLogNodeAddition21Node10Sec3NoSync.txt
Archive/Test results/APLogNodeAddition22Node10SecChStats(Multirun).txt
Archive/Test results/Discarded Tests
Archive/Test results/Discarded Tests/APLog21Node250Sec1AbsNoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLog21Node250Sec1NoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLog21Node250Sec2AbsNoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLog21Node250Sec2NoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLog21Node250Sec3AbsNoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLog21Node250Sec3NoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLog21Node250SecChStats(Multirun).txt
Archive/Test results/Discarded Tests/APLog22Node250SecChStats(Multirun).txt
Archive/Test results/Discarded Tests/APLog250sec21NodeMultiRun.txt
Archive/Test results/Discarded Tests/APLogNodeAddition20Node5Sec.txt
Archive/Test results/Discarded Tests/APLogNodeAddition21Node10Sec1NoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLogNodeAddition21Node10Sec2NoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLogNodeAddition21Node10Sec3NoChSwapNoSync.txt
Archive/Test results/Discarded Tests/APLogNodeAddition21Nodes10SecAbsNoChSwapNoSync(Multirun).txt

Table B.2.: Archived test results.

Archive/Test analysis
Archive/Test analysis/MultiAP-TrafficModel.xlsx
Archive/Test analysis/Testresults20Nodes10Sec1 - 16.05.13.xlsx
Archive/Test analysis/Testresults20Nodes10Sec2 - 16.05.13.xlsx
Archive/Test analysis/Testresults20Nodes10Sec3 - 16.05.13.xlsx
Archive/Test analysis/Testresults20Nodes10Sec4NoSync - 16.05.13.xlsx
Archive/Test analysis/Testresults21Nodes10Sec1NoSync - 20.05.13.xlsx
Archive/Test analysis/Testresults21Nodes10Sec1NoSyncAbsNoChSwap - 21.05.13.xlsx
Archive/Test analysis/Testresults21Nodes10Sec2NoSync - 20.05.13.xlsx
Archive/Test analysis/Testresults21Nodes10Sec3NoSync - 20.05.13.xlsx
Archive/Test analysis/Testresults22Nodes10secChStats(Multirun) - 23.05.13.xlsx
Archive/Test analysis/TestresultsChMgmt21Nodes15x200Sec1 - 20.05.13.xlsx
Archive/Test analysis/TestresultsChMgmt21Nodes15x200Sec2 - 21.05.13.xlsx
Archive/Test analysis/TestresultsChMgmt21Nodes15x200Sec3 - 21.05.13.xlsx
Archive/Test analysis/TestresultsChMgmt21Nodes15x200Sec4 - 22.05.13.xlsx
Archive/Test analysis/TestresultsChSwap21Nodes3APChSwapChange - 21.05.13.xlsx
Archive/Test analysis/TestresultsChSwap21NodesNoSync1 - 21.05.13.xlsx
Archive/Test analysis/TestresultsChSwap21NodesNoSync2 - 21.05.13.xlsx

Table B.3.: Archived test analysis spreadsheets.

Archive/2.4GHzRadioEvaluation.pdf
Archive/Calculations Period values and beacon size.xlsx
Archive/Channel Management Filter design.xlsx
Archive/Theoretical performance model.xlsx

Table B.4.: Other archived files and documents.