# Investigating Forward Error Correction Strategies on MSB430 Sensor Nodes

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Sebastian Samuel Barthlome
2011

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

# Contents

# List of Figures

# List of Tables

# Acknowledgment

I would like to thank my supervising tutor, Philipp Hurni, for being very helpful and patient during my work. Additionally, I appreciated the support of the other members of the RVS group, who provided helpful advices. Many thanks are dedicated to the developers of the RVS sensor testbed namely Philipp Hurni, Markus Anwander and Gerald Wagenknecht. Finally, a special thank goes to my girlfriend, Chantal Pfaffen, for attentively reading through my Master thesis.

Chapter 1

# Summary

The goal of this Master thesis is to investigate Forward Error Correction (FEC) on wireless, ultra low-power Modular Sensor Boards (MSB430) running the ScatterWeb$^2$ 1.1 Sensor Platform Operating System (ScatterWeb OS). Since wireless communication channels are prone to all kinds of wireless channel distortions, for example, due to multipath propagation, fading and scattering, bit errors occur with a much higher probability than in wired networks. One possible countermeasure against bit errors is the application of FEC on the transmitted packet payload. Bit errors occur as bit flips in transmitted packets and are caused by interferences. Without any FEC mechanism, corrupted packets are dropped by the receiver and the sender needs to retransmit the packet. Introducing FEC to the sender and the receiver allows to correct a certain amount of these bit errors on the receiver side.

The core component of FEC is an Error Correction Code (ECC). In this thesis, different ECCs are investigated, evaluated and compared with each other. The principle of each ECC is the same: On the sender node, there is an encoding component while the receiver contains the decoding component. The task of the encoder is to compute parity information over the bits to be protected. This parity information is appended to these bits and sent with them. On the receiver side, the decoder extracts the parity information and uses this information to correct bit errors if necessary. Not all ECCs are able to correct the same amount of errors. Moreover, the amount of parity information to correct the same amount of bit errors varies between the ECCs. In this thesis, the following ECCs were implemented in ScatterWeb OS and evaluated: The Repetition Code REP(3,8), the Hamming Code Hamming(7,4), the Double Error Correction Triple Error Detection Code DECTED(16,8), and the Bose-Hocquenghem-Chaudhuri Codes BCH(63,57), BCH(63,51), BCH(63,45), BCH(63,39), BCH(63,36). The experiments are gathered from different real-world topologies such as an indoor single link, an outdoor single link with line of sight, as well as from a multi-link topology deployed in the Institut für Informatik und angewandte Mathematik (IAM) building over several floors.

Since interferences are mostly short-lived and change rapidly, we developed three simple adaptive FEC mechanisms, which react to certain link quality changes. The three adaptive approaches use different information from past transmissions over a certain link as input to estimate the most appropriate for the upcoming transmission on the same link. Based on this input, the adaptive FEC mechanisms select an ECC from the list containing Hamming(7,4), DECTED(16,8), BCH(63,45), BCH(63,39), BCH(63,36) or simply apply no ECC. The simplest

approach is the Stateless Adaptive FEC (SA-FEC) approach. It just takes the last selected ECC into account. The Stateful Sender Adaptive FEC (SSA-FEC) approach is history-based and takes the used ECCs of the last five transmissions into account. The third and last approach is the Stateful Sender Receiver Adaptive FEC (SSRA-FEC) approach and is an extension of SSA-FEC. It selects an ECC based on the history of the used ECCs such as SSA-FEC does but additionally considers a history of the maximum number of corrected errors per code word. This information is delivered by the receiver in acknowledgement (ACK) messages. The selection of the history depending approaches is based on the moving average principle.

The results show that the selected ECCs are applicable to the MSB430 sensor nodes and worked on the ScatterWeb OS. Moreover, their application increases the amount of successfully delivered packets, especially for low quality links. The most frequently occurred errors in the payload of the packets are one or two bit errors. Therefore, the adaptive approaches mostly tend to select simpler ECCs, such as Hamming(7,4) and DECTED(16,8). The best results are achieved by adaptive FEC approaches.

The computational performance evaluation of the different ECCs conveys that the more complex ECCs such as the BCH codes need a lot more time for the decoding than for example the Hamming(7,4) code, whereas their corrective potential is rarely exploited to a full extent.

Chapter 2

# Introduction

In this Master thesis, we investigate and demonstrate the feasibility and potential of Forward Error Correction (FEC) mechanisms (see Section 2.2) on wireless ultra low-power Modular Sensor Boards (MSB430) (see Section 3.3.1). We show that with using FEC, the packet delivery rate (PDR) can be significantly increased. Besides eight Error Correction Codes (ECCs) (see Section 3.1), we implement three adaptive FEC schemes, which adapt to link quality changes at runtime.

A wireless sensor network (WSN) often consists of many wireless ultra low-power sensor nodes. The purpose of WSNs is to measure and detect environmental conditions or events, and trigger appropriate actions, e.g. triggering an alarm signal. The variety of possible sensed data is huge: For example motion and acceleration detection, sensing humidity, pressure, and temperature. Such measured sensor data is transported via the wireless links between the sensor nodes to a sink node for further processing. Besides the accuracy of the sensed data, a more important aspect is the reliability of the communication between the sensor nodes. It is crucial for the applications that the sensed data and the detected events are delivered quickly and reliable, since further actions depend on them.

A significant hazard for this reliability comes through signal interferences and distortions. Especially for WSNs deployed in buildings in urban areas, the communication inside the WSN can be seriously harmed by interferences with other signals. Such distortions can easily corrupt the packets exchanged between sensor nodes in such a way that the receiver is not able to interpret them anymore. Since decisions and triggering appropriate actions by the sink node may depend on information from the entire WSN, the loss of data caused by packet corruption can be crucial.

## 2.1 Automatic Repeat Request (ARQ)

The simplest and most naive way to deal with transmission errors is to retransmit the same packet again and again until it is received without errors or the maximum repetition value is reached. This behavior is described in RFC 3366 [5] and [6] in different Automatic Repeat reQuest (ARQ) protocols, that, besides wireless networks, are widely used in packet-switched networks. ARQ does not provide any mechanism to correct bit errors. ARQ can be used as

a standalone mechanism or can be combined with FEC as an additional option. In order to fulfill its task, an ARQ protocol needs a bidirectional communication between two nodes and a component, which checks the packet integrity. The functionality is simple: Before the sender sends a packet, it appends a cyclic redundancy check (CRC) [7] checksum to the packet. Then, the sender transmits the packet and waits for the ACK message from the receiver, which confirms the successful reception of the packet. In order to confirm the integrity of the packet, the receiver calculates the CRC checksum and compares it with the checksum that the sender added to the packet. If they match, the ACK message is sent from the receiver back to the sender. If the sender does not receive an ACK message from the receiver after a certain time, it assumes that the reception had failed and invokes a retransmission of the first packet. Among other reasons, the absence of an ACK message can be caused by a packet corruption of the original message. If the receiver receives a corrupted packet, the CRC checksum mismatches, which prevents the receiver to confirm the reception. There are other variants of ARQ such as the negative acknowledgment (NACK), where the receiver informs the sender about detected packet errors and not about successfully received packets. The case of applying ARQ as a standalone mechanism is not in the focus of this thesis, since it does not provide any bit error correction mechanisms.

## 2.2 Forward Error Correction (FEC)

A more sophisticated mechanism to overcome packet corruption is the concept of FEC [8]. FEC mechanisms are used in many electronic devices where bit corruption can happen during transmission over a noisy channel. FEC provides the ability to detect and correct a certain amount of bit errors in a bit stream. Like in the ARQ approach, FEC affects the sender as well as the receiver. Besides using a CRC checksum, the sender computes parity information over the data bits to be protected and appends this information to the data bits. A general term for the parity information is redundancy. Redundancy is defined as the additional information needed to enable the error detection and error correction [9]. The computation of the parity information is the task of the ECCs, that are discussed in Chapter 3. On the receiver side the same ECC is responsible to do an integrity check on the received data bits, which is achieved by looking into the parity information. Besides detecting errors, the ECC is able to correct some of the erroneous bits, which is basically the significant advantage compared to CRC. Since the sender adds the parity information in advance, the term “Forward Error Correction” has evolved. On one hand, the application of FEC always introduces a certain amount of overhead concerning the number of transmitted bits. On the other hand, if using FEC, the error correction happens immediately after the reception of a packet and the receiver does not have to wait for the retransmission like it is the case for ARQ.

FEC is not only used in wireless communication technologies but also in data storage systems. An example is a mass storage medium such as a DVD player where dust or scratches on the disk surface can lead to bit errors. Another example are ECC RAM modules mainly used in servers. In this thesis, we focus on the application of FEC in the environment of WSNs.

Figure 2.1 illustrates the principle of FEC applied to a sender and a receiver communicating over a wireless link. The sender contains a module called encoder using a certain ECC, which

computes the parity information. This parity information is then added to the information and builds a so-called code word. Wrapped into a packet, the code word is then transmitted to the receiver. The receiver uses the same ECC in reverse to decode, like the sender used to encode the data. The process consists of error detection, error correction and reassembling the original data.

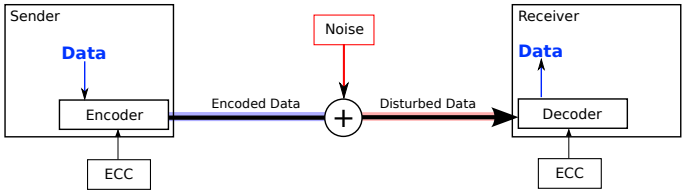



**Figure 2.1:** Encoder / Decoder Scheme

## 2.3 Adaptive Forward Error Correction

Wireless links are generally error prone due to interferences and signal distortions. Therefore, the quality of a wireless link can suddenly change and is very hard to predict. A decrease of the link quality can be caused by increasing interferences on the link or a decrease of the transmission power. In such a case, an appropriate reaction of the sender and the receiver is required. The sender needs to change to another ECC, which is more suitable to the current link quality. The selection of a more suitable ECC depends on the used adaptive FEC scheme and on the feedback of the receiver. If the receiver receives the packet correctly or fully restores the packet information, it sends a feedback message to the sender. Then, the sender considers to use a less powerful and simpler ECC for the next transmission on this link. Otherwise, if no feedback is received, the sender assumes that there were bit errors, which lead to packet corruption at the receiver side. In such a case, it needs to choose a more powerful ECC for retransmissions and upcoming transmissions on this link. In general, adaptive FEC mechanisms are summarized under the term of Hybrid Automatic Repeat reQuest (HARQ) [10].

Figure 2.2 presents a schematic picture of passing back the feedback from the receiver to the sender. The detailed description about the adaptive FEC mechanisms we implemented and how we choose the ECC which is used for a particular transaction are described in Chapter 3.

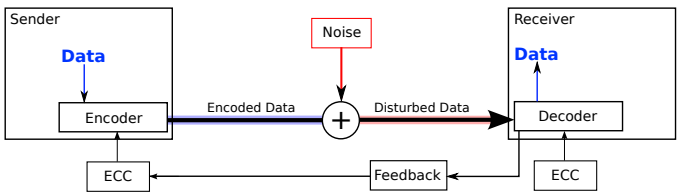



**Figure 2.2:** Adaptive Encoder / Decoder Scheme

## 2.4 Outline

In the context of this thesis, we consider static FEC mechanisms as well as a selection of adaptive FEC mechanisms. The following Chapter 3 briefly explains the implemented ECCs. Additionally, the sensor node platform as well as the ScatterWeb OS are described. Chapter 4 contains the description of the implementation of the ECCs. Moreover, the adaptive FEC approaches are explained. This chapter is finished by illustrating the challenges we faced. In Chapter 5, the results of the evaluation are discussed. Finally, Chapter 6 summarizes the conclusions and gives an outlook for future work.

# Chapter 3

# Related Work

In this chapter, the implemented Error Correction Codes (ECCs) are briefly portrayed. We present the advantages and disadvantages for each implemented ECC. We start with giving a short and general overview over ECCs and their characteristics. In the second last paragraph, we shortly summarize other similar studies applying FEC in the context of WSNs. Finally, the sensor hardware components and the key parts of the operating system are described.

## 3.1 Error Correction Codes (ECCs)

One of the pioneers in the research field of Error Correction Codes (ECCs) [11] was Richard Hamming. He was an American mathematician and invented one of the first ECC, the Hamming Code, in 1950 [12]. This simple code can be seen as the basis of the development of modern ECCs. The Hamming Code is described in Section 3.1.3.

### 3.1.1 Characteristics of ECCs

An ECC is the key part of any FEC mechanism. In general, there are two types of codes: Block codes and convolutional codes. Block codes take a predetermined amount of bits (a block) and encode it, while convolutional codes work on bit streams of arbitrary length. Historically, convolutional codes have been preferred because of the belief, that block codes could not be decoded efficiently. However, from the many years of research in the information theory, efficient decoding algorithms for linear block codes evolved. Block codes are also known to be memoryless, in the sense that the block to encode are treated independently from each other. In contrast, convolutional codes depend on the new information to encode as well as the already encoded information [11]. From our point of view this fact leads to more complex encoding algorithm, which may be inappropriate for sensor nodes. Therefore, in this Master thesis we focus on block codes.

ECCs basically do two things: They detect and correct bit errors. The number of detectable and correctable errors depends heavily on the particular ECC and characterizes its capabilities. Another characteristic of an ECC is how an encoded information is represented. If the original information is visible in an unchanged form in the encoded bit stream, we call it a systematic code, otherwise a non-systematic code. Systematic codes have one big advantage: decoding

is less complex than using a non-systematic code. In case of a systematic code in combination with no bit errors in the bit stream the decoding can be reduced to chop off the parity information from the received information. If we would have used a non-systematic code, we would need more operations to extract the original information from the encoded data. Since on MSB430 sensor boards only limited resources are available, we mainly use systematic block codes in this Master thesis.

It depends mainly on the trade-off between number of errors that can be corrected, and the complexity of the code, which ECC is applied. The complexity of an ECC depends on the computation time and the memory usage of the code. Moreover, it is crucial to consider how much additional information is added. The description of the implemented codes follows in the next paragraphs.

Since we implement these codes on ultra low-power sensors, we are forced to use simpler codes. Since the pure error detection capabilities of most ECC codes are still way below that of CRC, FEC are usually combined with CRC schemes.

### 3.1.2 Repetition Code

The Repetition Code [11] is one of the simplest and the most naive method to introduce error correction capability. This code is a good example for explaining how the process of encoding, correction and decoding works. The Repetition Code requires almost no computational power, but the overhead of the encoded information is very high. This ECC is well suited for low-power devices with small computational resources.

#### Encoder

The encoder of the Repetition Code takes $k$ bits (usually $k = 1$) and spreads each portion of $k$ bits by a given factor $r$, where $r$ is an odd number greater than one. This means that the length of the encoded information is multiplied by the factor $r$. For example, a Repetition Code with $k = 1$ and $r = 3$, abbreviated by REP(3,1), encodes a message $m_{orig} = 10010$ in the code word $m_{enc} = 111000000111000$. As $m_{enc}$ shows, a 1-bit is spread to $111$ and a 0-bit to $000$.

#### Error Detection and Correction

The Repetition Code provides only a very naive way of error detection and error correction. These operations are both integrated into one operation. It takes the parameters $k$ and $r$ as input, which were used to encode a message. These parameters define how many bits are used to encode one single bit and where to find the corresponding bits for a single bit of the original message. The operation consists of counting the number of the $r$ 1-bits or 0-bits in the encoded message that represent the corresponding bit in the original message. The condition for $r$ to be an odd number guarantees that there is no possibility for the 0-bits counter and the 1-bits counter to be equal. The Repetition Code is simple and fast but introduces an overhead in encoded data of a factor $r$. The error correction capability of the REP(3,1) code is limited to a single bit error.

### Decoder

The Repetition Code decoder inserts either a 1-bit or a 0-bit depending on which counter was higher. The detection and the correction operation are integrated in the decoder. The Repetition code does not correct bit errors in the received message in terms of swapping bits in the message actually. It reads the received message and estimates the original message depending on the bit counters for each bit. The estimation is based on the assumption that the more frequent a bit value occurs, the higher is the probability that the original bit has this value. This procedure is called Majority Logic Decoding [13]. The advantages and disadvantages of the Repetition Code are listed in Table 3.1.

| Advantages | | Disadvantages | |
|---|---|---|---|
| + | Easy to implement | - | Low error correction |
| + | Fast encoding and decoding | - | High transmission overhead |
| + | Little computation power usage | | |

**Table 3.1:** Advantages and Disadvantages of Repetition Code

## 3.1.3 Hamming Code

Hamming Codes belong to the class of linear block codes. The general notation of Hamming Codes is Hamming($n$,$k$). The length of an encoded word $n$ is computed as $n = 2^m - 1$, where $m = n - k$ is the number of parity bits. The number of data bits that are encoded in one code word is $k = 2^m - m - 1$. The mathematical construct of a Hamming Code contains two binary matrices: The Generator matrix $\mathbf{G}$, whose rows are linearly independent

$$\mathbf{G}_{k,n} := (I_k| - A^T) = \left[\begin{array}{cccc|cccc} i_{11} & & & & -a_{11} & -a_{21} & \cdots & -a_{n-k1} \\ & i_{22} & & & -a_{12} & -a_{22} & \cdots & -a_{n-k2} \\ & & \ddots & & \vdots & \vdots & \ddots & \vdots \\ & & & i_{kk} & -a_{1k} & -a_{2k} & \cdots & -a_{n-kk} \end{array}\right]$$

and the Parity check matrix $\mathbf{H}$, whose rows are linearly independent

$$\mathbf{H}_{n-k,n} := (A|I_{n-k}) = \left[\begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{1k} & i_{11} & & & \\ a_{21} & a_{22} & \cdots & a_{2k} & & i_{22} & & \\ \vdots & \vdots & \ddots & \vdots & & & \ddots & \\ a_{n-k1} & a_{n-k2} & \cdots & a_{n-kk} & & & & i_{n-kn-k} \end{array}\right]$$

For $\mathbf{G}$ and $\mathbf{H}$ the following condition has to be fulfilled (see [14]):

$$
\begin{aligned}
\mathbf{HG}^T &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} & i_{11} & & & \\ a_{21} & a_{22} & \cdots & a_{2k} & & i_{22} & & \\ \vdots & \vdots & \ddots & \vdots & & & \ddots & \\ a_{n-k1} & a_{n-k2} & \cdots & a_{n-kk} & & & & i_{n-kn-k} \end{bmatrix} \\
&\cdot \begin{bmatrix} i_{11} & & & & & & \\ & i_{22} & & & & & \\ & & \ddots & & & & \\ & & & i_{kk} & & & \\ -a_{11} & -a_{12} & \cdots & -a_{1k} & & & \\ -a_{21} & -a_{22} & \cdots & -a_{2k} & & & \\ \vdots & \vdots & \ddots & \vdots & & & \\ -a_{n-k1} & -a_{n-k2} & \cdots & -a_{n-kk} & & & \end{bmatrix} \\
&= \begin{bmatrix} a_{11} - a_{11} & a_{12} - a_{12} & \cdots & a_{1k} - a_{1k} \\ a_{21} - a_{21} & a_{22} - a_{22} & \cdots & a_{1k} - a_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-kn-k} - a_{n-kn-k} & a_{n-k2} - a_{n-k2} & \cdots & a_{n-kk} - a_{n-kk} \end{bmatrix} \\
&= \mathbf{A} - \mathbf{A} \\
&= 0
\end{aligned}
$$

The matrix $\mathbf{A}$ which is the common part of $\mathbf{G}$ and $\mathbf{H}$ represents the coverage of the parity bits over the data bits. The key idea of selecting the parity bits is to cover each data bit with a different unique set of parity bits. The other element in the matrices $\mathbf{G}$ and $\mathbf{H}$ is the Identity matrix $\mathbf{I_k}$ and the Identity matrix $\mathbf{I_{n-k}}$, respectively. Since we use binary numbers, all the matrix operations are computed in the Galois Field GF(2) using modulus 2. This means that $1 + 1 = 1 - 1 = 0$.

The formulas above define the relation between the parameters that define a Hamming Code. Detailed instructions for the construction of Hamming Codes can be found in [15]. Due to practical reasons, we used the ECC Hamming(7,4). It encodes 1 byte (8 bits) into 14 bits, which fit into a 2-bytes array. The encoder and decoder used in the example of a Hamming(7,4) code is based on [15] and briefly explained in the following paragraphs. The Hamming(7,4) code is a so-called Single Error Correction Double Error Detection (SECDED) Code. There are extensions of Hamming Codes which correct multiple errors per code word.

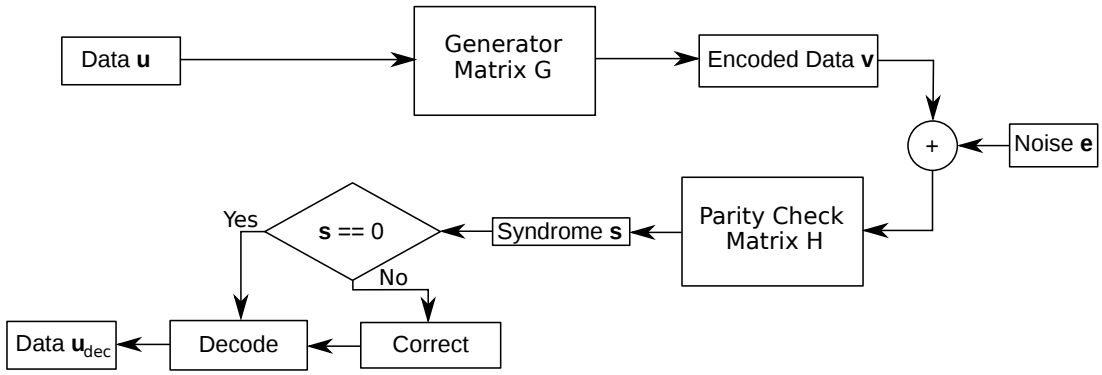



**Figure 3.1:** Encode / Correct / Decode in Hamming Code

### Encoder

The encoding process uses the Generator matrix $\mathbf{G}_{4,7}$.

$$\mathbf{G}_{4,7} := \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$\mathbf{G}$ has the identity matrix $\mathbf{I}_4$ on the left side in the first four columns. The second three columns lists $-A^T$. Under the assumption of four data bits $d_0, d_1, ..., d_3$ we need three parity bits $p_0, p_1, p_2$ to cover each data bit $d_i$ with an unique set of parity bits. Comparing the information in Table 3.2 with the right side of $\mathbf{G}$, one notices that this is the information about which data bit is covered by which parity bit. Figure 3.2 illustrates this coverage. For example, the data bit $d_0$ is covered by the parity bits $p_0$, $p_1$, etc.

| | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
|---|---|---|---|---|
| $p_0$ | 1 | 1 | 0 | 1 |
| $p_1$ | 1 | 0 | 1 | 1 |
| $p_2$ | 0 | 1 | 1 | 1 |

**Table 3.2:** Covering of Data and Parity Bits

The actual code word $\vec{v}$ is computed as $\vec{v} = \vec{u}\mathbf{G}$, where $\vec{u} = u_0u_1...u_{k-1}$ are the data bits to be encoded. This results in a code word $\vec{v}$ containing $n$ bits. Let us assume that we want to encode 1 byte, that contains the bits 10010101. Since we can only put $k = 4$ bits into one block, the byte needs to be split into $\vec{u_0} = 1001$ and $\vec{u_1} = 0101$. These two blocks are encoded separately to the code words $\vec{v_0} = \vec{u_0}\mathbf{G} = 1001001$ and $\vec{v_1} = \vec{u_1}\mathbf{G} = 0101010$, where the green bits are parity information. The result of encoding the byte 10010101 leads to $\vec{v_{01}} = \vec{v_0}\vec{v_1} = 10010010101010$.

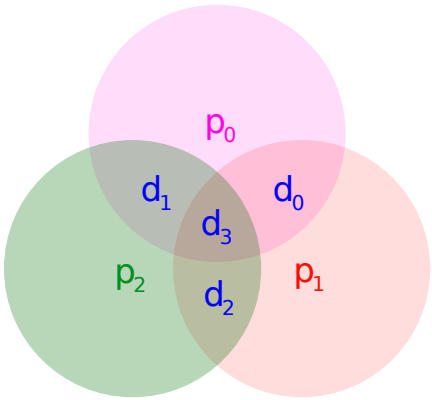



**Figure 3.2:** Parity / Data Bit Covering [1]

As described in Section 3.1, there are systematic and non-systematic ECCs. Since in this example we used a systematic ECC, the first $k$ bits of our code word $\vec{v}$ are identical to the original data bits that has been encoded. The rest of the bits are the $m$ computed parity bits. If we want to generate a non-systematic Hamming Code, we just need to swap the columns in $\mathbf{G}$ and create the corresponding Parity check matrix $\mathbf{H}$. This will then generate a non-systematic code word.

### Error Detection and Correction

The error correction and detection capability of a Hamming Code depends on the so-called minimum Hamming distance $d_{min}$. It defines the minimum number of bit substitutions to transform one valid code word into another code word of the same Hamming Code. The minimum distance $d_{min}$ is consequently the smallest distance between any two code words of a single Hamming Code. The error correcting capability $t$ is defined as follows:

$$t = \lfloor (d_{min} - 1)/2 \rfloor$$

In our example of the Hamming(7,4) code $d_{min}$ is equal to three. Therefore, this ECC is able to correct one single bit error per code word. As shown in Figure 3.1, the encoded word which may contain some bit errors, is processed by the Parity check matrix $\mathbf{H}$. It is constructed by listing all columns of length $m$ that are pair-wise independent. The Parity check matrix $\mathbf{H}$ in our example has the following form:

$$\mathbf{H}_{3,7} := \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The computation $\vec{s} = \vec{r}\mathbf{H}^\mathbf{T} = (\vec{v} + \vec{e})\mathbf{H}^\mathbf{T} = \vec{u}\mathbf{G}\mathbf{H}^\mathbf{T} + \vec{e}\mathbf{H}^\mathbf{T} = \vec{e}\mathbf{H}^\mathbf{T}$ results in a so-called syndrome vector $\vec{s}$. The vector $\vec{e}$ models the bit error positions introduced to $\vec{v}$. The vector $\vec{r}$ is the received encoded word and can be modeled as $\vec{e}+\vec{v}$. Because of the precondition $\mathbf{H}\mathbf{G}^T = 0$,

it follows that $\mathbf{G}\mathbf{H}^T = 0$ and the syndrome vector only depends on the error vector $\vec{e}$. The syndrome vector is reduced to $\vec{s} = \vec{e}\mathbf{H}^\mathbf{T}$. It has the length $m$ and contains the information about the location of the bit error. If all values of the vector are equal to zero, no bit error occurred. In this case, no error has to be corrected and we can go directly to the decoding part.

In case the syndrome is not zero, errors need to be corrected. The bits which are equal to one indicate, which bits were flipped and need to be corrected. The error positions can be matched to an unique combination of 1-bits in the syndrome vector.

Considering the example presented in the previous section, it is assumed that the decoder receives $\vec{r_{01}} = \vec{r_0}\vec{r_1} = 10110010101010$ with a bit error at the marked position. The computation of the syndrome vector $\vec{s_0}$ of $\vec{r_0}$ leads to a non-zero syndrome $\vec{s_0} = \vec{r_0}\mathbf{H}^\mathbf{T} = 011$ that indicates the error position. Doing the same computation for $\vec{r_1}$ results in the syndrome vector $\vec{s_1} = \vec{r_1}\mathbf{H}^\mathbf{T} = 000$. This means that $\vec{r_1}$ contains no bit error. After completing the correction process our bit stream looks like $\vec{r_{corr_{01}}} = 10010010101010$.

### Decoder

Since in this example a systematic ECC is used, the decoding process is very easy and can be done fast without complex computations. The only step to be done is to chop off the part with parity bits. In the example above, simply the green parity bits from each received code word $\vec{r_{corr_0}} = 1001001$ and $\vec{r_1} = 0101010$ need to be chopped off. To retrieve our original byte we encoded in the beginning of the example, both decoded code words are merged together $\vec{u_{dec_{01}}} = \vec{u_{dec_0}}\vec{u_{dec_1}} = 10010101$. Table 3.3 lists the advantages and disadvantages of the Hamming ECC.

| Advantages | Disadvantages |
|---|---|
| + Simple encoding | - High amount of parity |
| + Linear operations | |
| + Fast encoding | |
| + Less amount of redundancy than Repetition Code | |

**Table 3.3:** Advantages and Disadvantages of Hamming Code

## 3.1.4 Double Error Correction Triple Error Detection

One version of a Double Error Correction Triple Error Detection (DECTED) code was proposed by [16]. It is called DECTED(16,8). This code is able to correct up to two bit errors and can detect triple-adjacent bit errors. Just like Hamming codes, DECTED(16,8) is a block code. It works very similar as the Hamming Code. The DECTED(16,8) code takes 8 bits input and creates an encoded word of 16 bits out of it. The encoded word consists of the original data block in the first byte (bits 0 to 7) and the second byte (bits 8 to 15) contains the parity bits. This situation is convenient because they fit perfectly into two bytes in the memory.

### Encoder

The entire process is illustrated in Figure 3.1. The encoder unit takes 8 bits and computes the encoded word by using the Generator matrix $\mathbf{G}$, as presented in [16]. The Generator matrix $\mathbf{G} := (I_8|C^T)$ used here has the following form:

$$\mathbf{G}_{8,16} := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The encoded word is retrieved by computing $\vec{v} = \vec{u}\mathbf{G}$, where $\vec{u} = u_0u_1u_2...u_7$ contains the 8 bits to encode and $\vec{v} = v_0v_1v_2...v_{15}$ represent the 16 bits of the corresponding encoded word. As an example, the encoding of the byte $\vec{u} = u_0u_1u_2...u_7 = 10010101$ is demonstrated. Computing the code word leads to $\vec{v} = \vec{u}\mathbf{G} = 1001010110100000$ with the green bits as parity information.

### Error Detection and Correction

In order to detect and correct bit errors, a syndrome vector $\vec{s}$ is calculated using the corresponding Parity check matrix $\mathbf{H}$. It is defined as $\mathbf{H} := (C^T|I_8)$ and has the following form:

$$\mathbf{H}_{8,16} := \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The received encoded word $\vec{r}$ can be modeled as $\vec{r} = \vec{v} + \vec{e}$, where $\vec{v}$ is the encoded word and $\vec{e}$ the error vector. Because of the construction of the matrices $\mathbf{G}$ and $\mathbf{H}$, every encoded word $\vec{v}$ is orthogonal to the rows of $\mathbf{H}$. This leads to $\vec{v}\mathbf{H^T} = 0$, which means that the syndrome vector $\vec{s} = \vec{r}\mathbf{H^T} = \vec{e}\mathbf{H^T}$ only depends on the error vector $\vec{e}$. As for the Hamming Code, the syndrome vector contains the information of the faulty bits. If a syndrome vector only contains zeros there are no errors in the received code word $\vec{r}$. All other syndrome vectors indicate that there are errors in the received code word. A syndrome vector not equal to zero is unique and identifies a bit error pattern consisting of a one bit error or a two bit error. Therefore, it can be uniquely determined which bits were flipped and need to be corrected.

We assume that during transmission of the encoded byte $\vec{r} = 1011010110110000$ the two red bits are flipped. To detect the errors, the syndrome vector has to be computed as follows: $\vec{s} = \vec{r}\mathbf{H}^{\mathbf{T}} = 10001010$. Since this syndrome vector is not zero, there are errors to correct. The two bit errors in the example can only occur in the following combinations: Both of the errors are in the parity bits, both errors are in the data bits, or there is one error in the data bits and one in the parity bits. If both errors are in the data bits, the syndrome vector corresponds to a unique combination of two columns of the matrix $\mathbf{C}^{\mathbf{T}}$. In our example, this is not the case. If both errors are in the parity bits, an unique combination of the columns of the matrix $\mathbf{I_8}$ would be equal to the syndrome vector. Neither of this is the case in our example. From this follows that there is one error in the data bits and one in the parity bits. Syndrome vectors indicating such two bit errors, correspond to the unique combination of a column of $\mathbf{I_8}$ and a column of $\mathbf{C}^{\mathbf{T}}$. If the columns of $\mathbf{I_8}$ and $\mathbf{C}^{\mathbf{T}}$ are counted from left to right, one can observe that this combination consists of the third column of $\mathbf{C}^{\mathbf{T}}$ and of the fourth column of $\mathbf{I_8}$ and corresponds to the computed syndrome vector. The index of the columns now indicate where the faulty bits are located in the code word. As one can see, the detection process found the red bits as the corrupted bits. They are at the third position of the data bits, and the fourth position of the parity bits reading the bits from left to right. The details about the detection and correction of single bit errors an the other double bit errors are not presented here, but can be found in [16].

### Decoder

Because of the construction of the Generator matrix $\mathbf{H}$, the DECTED(16,8) code generates a code word which includes the original information in one continuous bit sequence. This fact identifies the DECTED(16,8) ECC as a systematic code. The decoding process of the DECTED(16,8) code is straightforward. We only need to ignore the second byte which only contains parity information in order to retrieve the original information. The advantages and disadvantages of the DECTED(16,8) ECC are listed in Table 3.4.

Since the errors are corrected successfully in the given example, only one step is left to do. The decoder just needs to chop off the green parity bits from the corrected code word $\vec{r_{corr}} = 1001010110100000$. After finishing the decoding process, we obtain the decoded byte correctly as $\vec{u_{dec}} = 10010101$.

| Advantages | Disadvantages |
|---|---|
| + Fits well into bytes | - Adds 100% parity |
| + Corrects up to double bit errors | - Large matrices to store |
| + Linear operations | |
| + Fast | |

**Table 3.4:** Advantages and Disadvantages of DECTED Code

### 3.1.5 BCH Code

A good choice to correct multiple bit errors is to consider Bose-Hocquenghem-Chaudhuri (BCH) codes. BCH codes were invented in the 1959 by Hocquenghem [17] and independently in 1960 by Bose and Ray-Chaudhuri [18]. These ECCs belong to the class of cyclic block codes, where every shift of a code word is another code word generated by the same code. Another important characterization of BCH codes is that they can be constructed in such a way that any number of bit errors in a code word of a given length can be corrected. Cyclic ECCs are used for example in CD-ROM drives, where errors occur through scratches in the surface of the disc or dust in the drive. BCH codes are used especially in the Digital Video Broadcasting – Terrestrial (DVB-T) standard. A flow chart of the operations of the BCH is shown in Figure 3.3. In the following paragraphs the general principle of the BCH codes is briefly explained. BCH codes are defined by two parameters: $n$ and $k$. The BCH(n,k) code encodes $k$ bits into a code word of length $n$ bits.

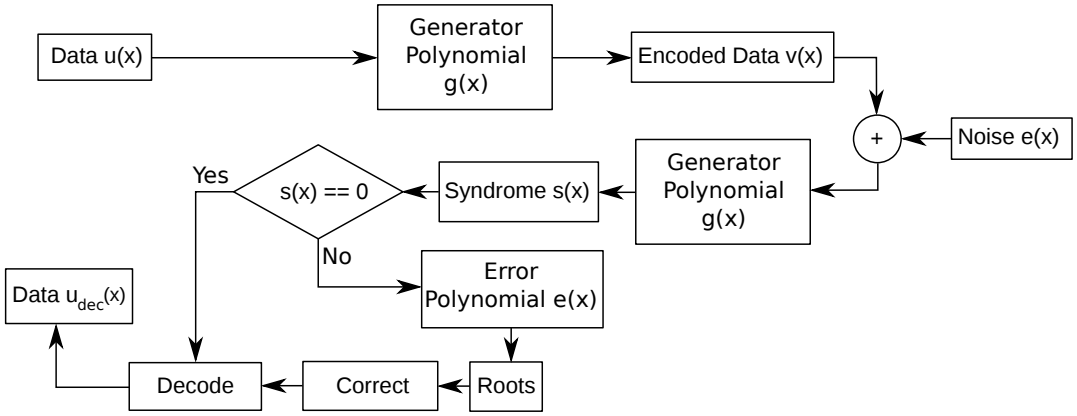



**Figure 3.3:** Encode / Correct / Decode in BCH Code

#### Encoder

The key part of the encoding unit is the so-called Generator polynomial $g(x)$. Like the Generator matrix in the Hamming Code, the Generator polynomial plays the same role in the BCH code. A block of data to be encoded is processed by the Generator polynomial. We do not go into mathematical details of the construction of the Generator polynomial here. A detailed description can be found in [11] [10]. The Generator polynomial has the same role as a prime number. Each code word can be represented as the sum of multiple products of such a generator polynomial. Therefore, it divides every code word produced by this generator polynomial. This means that the generator polynomial consists of irreducible factors, which are also polynomials. All computations are done in the corresponding Galois Field (GF) and the bit sequences are represented as polynomials in this field. To generate the code word out of a sequence of bits, we use a method based on modulo division, leading to a systematic code. Suppose we have a sequence of bits $u$ to encode and the polynomial representation of it is $u(x)$. The code word

$v(x)$ is then computed as $v(x) = u(x) \cdot x^k - ((u(x) \cdot x^k) \bmod g(x))$, where $u(x) \cdot x^k$ is a shift of the original bit sequence of $k$ positions, where $k$ is the degree of the Generator polynomial. This shift of the original bits creates space for the rest of the modulo division and generates a systematic code word.

### Error Detection and Correction

The error detection is achieved by computation of a syndrome polynomial $s(x)$. Assume the received code word is $r(x) = v(x) + e(x)$, where $e(x)$ is the unknown error polynomial.

The syndrome polynomial $s(x)$ is then computed as $s(x) = r(x) \bmod g(x) = e(x) \bmod g(x)$. If $s(x) = 0$, the transmission was error-free. Otherwise, the error correction procedure is invoked. In a first step, the determination of flipped bits in the received code word needs to be done. In other words, the error polynomial $e(x)$ has to be found. The Berlekamp-Massey algorithm [19, 20] invented by Elwyn Berlekamp and James Massey does this task. The Berlekamp-Massey algorithm takes the syndrome polynomial $s(x)$ as input and returns the error polynomial $e(x)$. This polynomial contains the information about the errors. In the second step, the determination of the error locations is done. This is achieved by searching the unique roots of the error polynomial, which is the task of the Chien Search algorithm [21]. The input for the Chien Search algorithm is the error polynomial $e(x)$ and it returns the roots of the polynomial which correspond to the error positions. The final step is to evaluate the error value and correct it at the given positions. Because of the binary number system, the error value is 1. This means that the erroneous bits just need to be flipped. For the mathematical details of the entire correction process see [22].

### Decoder

Like for all systematic ECCs, it is very easy to decode a code word if no error has occurred. We simply need to get rid of the appended part containing the rest of the modulo division. This is done by shifting the original information to its origin. This corresponds to an inversion of the part $u(x) \cdot x^k$ of the encoding procedure. The advantages and disadvantages of the BCH ECCs are listed in Table 3.5.

| Advantages | Disadvantages |
|---|---|
| + Low amount of redundancy | - Complexity |
| + Easy to implement in hardware | - Iterative and complex decoding algorithm |
| + Widely used | |

**Table 3.5:** Advantages and Disadvantages of BCH Code

## 3.2 Error Patterns and FEC Schemes in WSNs

The topic of applying FEC in WSNs has yet been investigated in a couple of studies. ECCs in WSNs were investigated on different hardware platforms with different radio modules such as the Chipcon CC1000 [23], or the RFM TR 1001 [24]. It is a general observation that the application of FEC increases the reliability on wireless links. Although the feasibility of powerful and complex ECCs is difficult on the resource limited sensor nodes.

The experimental topologies and scenarios used for the evaluation differ among the related publications. Often, the applied topologies seem to be far away from real-world topologies. For example the topology used by Busse et al. in [25] consisting of 16 sensor nodes in a line of sight and one sender. With this evaluation, important crucial wireless phenomena are not taken into account at all, e.g. signal distortions through concrete walls and floors. Another example is the topology described by Willig et al. [24], where the distance between any two nodes was fixed to 30 cm. Some studies limited their experiments only to network simulators under simplistic channel assumptions, e.g. ns-2 in [26].

Jeong et al. [23] described that the most occurring errors in a close to real-world, indoor experiment are single-bit and double-bit errors. Therefore, a valid conclusion is that complex but powerful ECCs are not necessarily required especially considering the resource usage of complex ECCs. This conclusion is also supported by Busse et al. [25], who additionally argued with the simplicity of the implementation of the ECCs and the requirements concerning the power of the error correction of applications. Jeong et al. also stated that the selection of an appropriate ECC depends on the application for which a WSN is designed, since energy consumption and resource usage by the applied ECC has a huge impact on the lifetime of the WSNs.

Particularly, application of WSNs demand energy concerned error control mechanisms. On one hand, ECCs are required concerning the reliability of the communication and on the other hand, the energy consumption of the sensor nodes need to be considered. Willig et al. related error control mechanisms with the energy consumption [24]. The understanding of the channel error patterns is crucial to design energy-efficient error control schemes. They also state that in a early design state of such schemes, theoretical error models can be helpful. However, empirical measurements are essential to quantify the energy costs of error control schemes. Hence, the study also evaluated some hybrid FEC/ARQ strategies in a simulation environment.

Busse et al. tackled the idea of interleaving code words to cope with burst errors. Applying code word interleaving may helps to divide the impact of burst errors up on several code words which can be corrected by simpler ECCs. They concluded that for some ECCs, the application of code word interleaving in combination with FEC is an advantage concerning the error correction.

### Adaptive FEC Mechanisms

In general, it is hard to predict the frequency and severity of signal distortions. Therefore, it is almost impossible to choose the right ECCs in advance at compile-time. This immediately leads to the question whether adaptive FEC mechanisms could be a valid alternative. Simulation based experiments with adaptive FEC mechanisms were presented by Ahn et al. and Willig et al. [26] [24]. Real-world experiences with adaptive schemes in WSNs have not yet been studied.

So-called FEC-level adaptations (FECA) for wireless networks were proposed in [26], which only adapts the parameters of an ECC, such as the amount of parity bits. FECA is based on the so-called type-I hybrid ARQ approach. It retransmits the packet together with parity bits in contrast to the type-II hybrid ARQ approach, where only parity bits are retransmitted. This approach needs the receiver to buffer the previous packet. A mobile device using the FECA approach does not depend on a explicit feedback information. For an appropriate adaptation of the FEC strength to the channel state transitions, FECA expedites upward and downward transitions to the higher and lower level of FEC strength. Such transitions are activated by either a packet loss or the timeout of an backoff timer. FECA is not explicitly designed for wireless ultra low-power sensor nodes but rather for 802.11 wireless networks. The experiments were performed in the network simulator ns-2 using a generic integrated error model for the wireless channel. They concluded that FECA performs better than static FEC mechanisms, given that the error rates do not oscillate too rapidly.

For years, the majority of studies investigating wireless (sensor) networks were based on simulations. However, a discrepancy between the results of simulation-based results and results from real-world measurements has been more and more questioned. Especially in WSNs and the ad-hoc community, simulation-based measurements have been identified as a general drawback due to inappropriate parameter settings and unrealistic radio, traffic and error models [27] [28]. The trend in the research field of wireless communication heads to proof the feasibility of the proposed mechanisms and protocols on real-world devices. Therefore, we focus in this thesis on real-world experiments. For the evaluation of the protocols and mechanisms in real-world topologies, experimental sensor network testbeds have become indispensable.

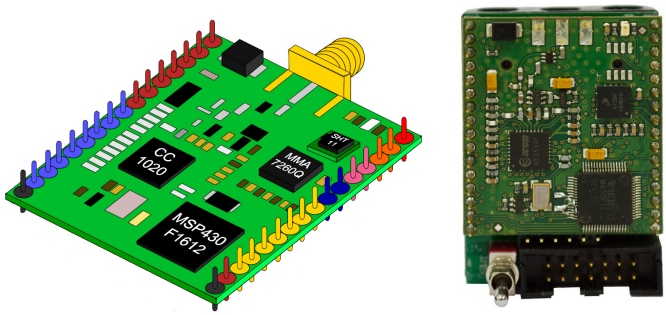
This thesis further distinguishes from the related work in the field of WSNs by the evaluation of eight different ECCs, ranging from simple codecs with a low correctional power to very sophisticated codes with an extended error correction capability. Moreover, some of our adaptive FEC mechanisms are history-based as opposed to the yet studied approaches, taking several packet exchange processes into account. Additionally, our approaches work in a hop-by-hop manner, which allows to react more precise to local link interferences independently from the entire network.

## 3.3 Sensor Nodes

### 3.3.1 MSB430

This section briefly presents the hardware platform used throughout this thesis. We used the ultra low-power Modular Sensor Board MSB430 (MSB430) [29] developed and distributed by ScatterWeb GmbH [30]. It is a modular research platform featuring a MSP430 series RISC CPU (MSP430F1612) from Texas Instruments, a CC1020 fully software configurable wireless radio chip from Chipcon, an external SD card reader, and two sensors (SHT11, MMA7260Q). The MSP430F1612 is clocked with 100 kHz up to 11 MHz. The clock speed can be adapted by a software configurable digital oscillator (DCO). The MSB430 has 55 KB flash memory and 5 KB RAM. The CC1020 transceiver uses a low-noise amplifier and operates in the ISM-band on the frequency of 868 MHz. Its output power reaches an amplitude up to 8.6 dBm (7.2 mW). The CC1020 uses 8 channels with a data rate of 19.2 kbit/s when using Manchester encoding. The two sensors that are onboard are the Sensirion SHT11 sensor, which senses temperature and relative humidity and the sensor MMA7260Q from Freescale, which measures the 3D-acceleration. The external Secure Digital (SD) card slot supports Secure Digital High-Capacity (SDHC) cards with a capacity up to 32 GB. The MSB430 has 18-digital I/O pins which are connected to a analog-to-digital (ADC) and digital-to-analog (DCA) converters. The datasheets of the used main components on the MSB430 and the circuit diagrams of the mainboard can be found in [3].

For the connection with the computer two connectors are on-site. First, the JTAG connector is used to program the firmware of the MSB430. Second, there is an UART interface through which the MSB430 is connected to a PC using a FTDI converter cable. The UART interface can be accessed by a terminal and is used for debugging and logging. The power for the MSB430 can be taken from three AAA batteries mounted under the board or via the FTDI converter cable from the computers USB connector. An onboard switch changes between these two power sources.




(a) MSB430 Scheme (b) MSB430 Node

**Figure 3.4:** MSB430 Modular Sensor Board [2] [3]

### 3.3.2 ScatterWeb$^2$ 1.1 Sensor Platform Operating System

We implemented the selected ECCs in ScatterWeb$^2$ 1.1 Sensor Platform Operating System (ScatterWeb OS) [30] developed at the TU Berlin in 2007. This modular, event-driven, single-threaded OS written in C provides libraries for operating the CPU, radio module, sensor event handling, and storage management. It is designed in a lightweight way to operate and manage the MSB430 sensor nodes (see Section 3.3.1). The ScatterWeb OS runs a superloop, which can be interrupted by events. These events are then processed by so-called event handlers that cannot be preempted by other event handlers and run until completion. Such events trigger interrupts and force the program to deal with it. Examples for interrupt triggering events are sensor event detection, incoming and outgoing transmissions, timers, and communication over the serial interface of the sensor node. In contrast to traditional desktop operating systems, the ScatterWeb OS kernel is not placed in a separated memory block distinct from memory blocks where the application runs. This circumstance can lead to kernel panics, when a program modifies memory used by the kernel. Another difference to desktop operating systems is that the ScatterWeb OS is not multi-threaded. There is simply not enough memory available on the senor nodes to provide every blocked thread its own stack, as it is the case in multi-threaded systems.

The user interface to the ScatterWeb OS is provided through commands that are entered over a serial communication interface through a terminal client running on the computer to which the sensor is connected. In order to prevent the ScatterWeb OS to be stuck, it has a mechanism called Watchdog. This mechanism counts the clock ticks needed for executing a certain part of the program. If it exceeds a certain limit, it raises a panic message known as Watchdog panic. The ScatterWeb OS implements a mechanism to perform delayed actions, so-called software timers. This mechanism uses timers that can be scheduled in a queue with a certain delay. The system then executes the earliest scheduled timer. If a timer is executed, it simply calls a predefined function. The structure of the ScatterWeb OS is shown in Figure 3.5. It contains the most important components including the ECC library, that we implemented. The ECC library is not part of the general ScatterWeb OS. In the next paragraph, we shortly explain the main and relevant components we touched by our implementation.

#### ScatterWeb Components

The system core contains the OS kernel modules. These modules build the basis to perform any action on the senor node. While booting the sensor node, the module System.Boot.c loads and configures the modules needed for operation. This includes setting up the hardware relevant modules such as the radio module (CC1020). After the boot process is finished, the node starts to execute the described superloop. It is implemented in the System.c module, which is the core of the ScatterWeb OS. To be able to schedule timers, a module that administrates the timers is required. This is done in the module called Timers.c. It contains and manages the queue containing the scheduled timers. It tells the superloop to process the timer when it should be executed.

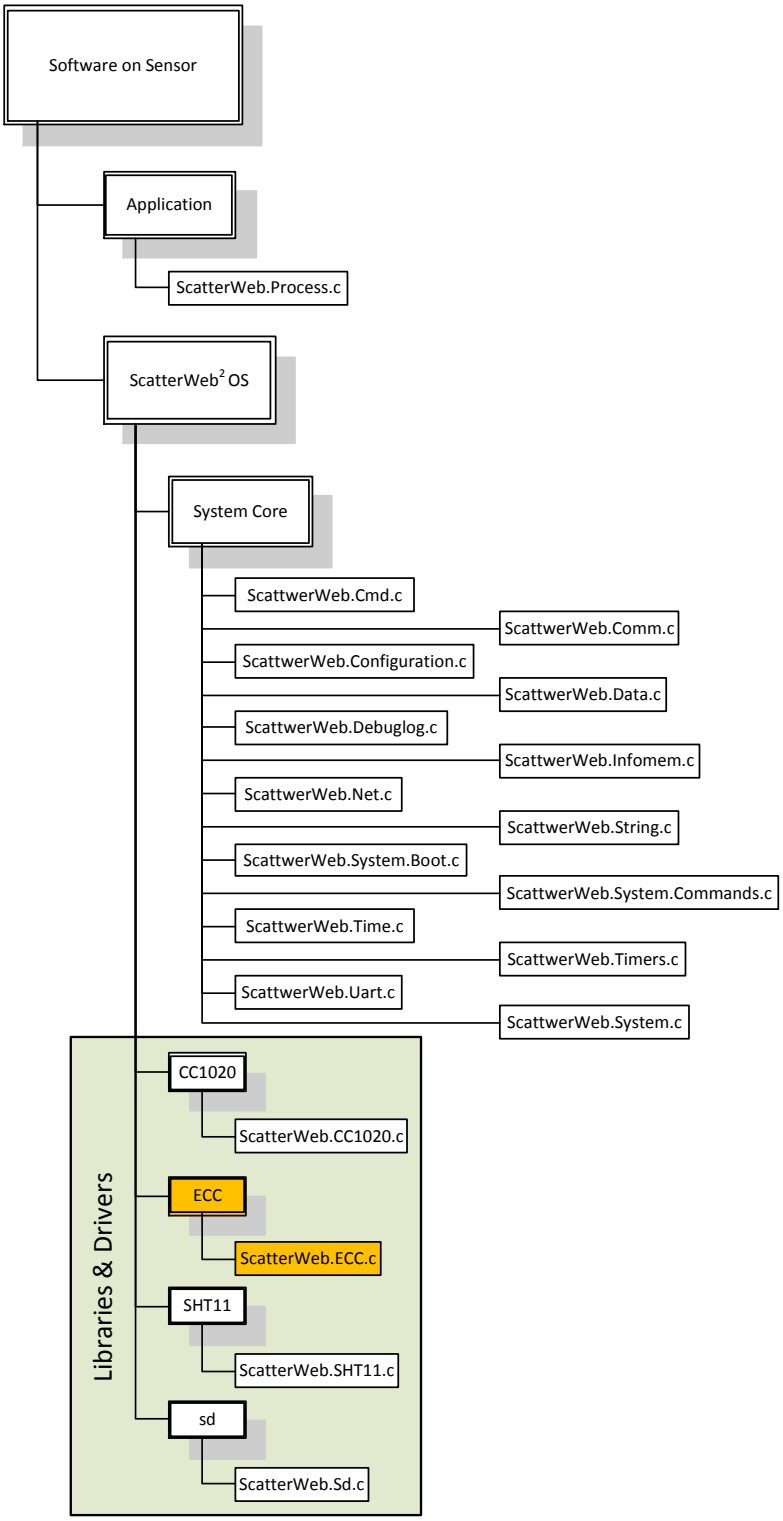



**Figure 3.5:** ScatterWeb OS Software Components

The ScatterWeb OS supports to store persistent configurations. This means that certain parameters can be configured and are available after a restart of the node, even if it has been disconnect from power before rebooting. The module responsible for this is the Configuration.c module. It writes and reads the configuration to and from the flash memory. If a node is started, the configuration is automatically read from memory, and the system is configured according to it.

Another important module is the network module implemented in Net.c. It provides an interface for ScatterWeb OS and its applications to the radio module (CC1020). The network module contains methods for handling packets to be send and received. A packet to be sent is taken from an application or from the ScatterWeb OS itself and the network module coordinates the transmission of it. It contains queues for incoming and outgoing packets. Moreover, it is checking whether there is a transmission ongoing that is determined for this node or not. For our implementation, the network module was the most affected standard ScatterWeb OS module. More details on the implementation of the ECC and the changes made in the network module can be found in Section 4.1.

The module between the network module and the radio chip is the CC1020 radio driver module. It is the driver for the radio chip and provides methods to turn on/off and calibrate the radio chip.

While the module SHT11.c provides control of the sensor SHT11, the SD module (Sd.c) gives access to the external SD card slot.

Chapter 4

# Implementation of Error Correction Codes

In this chapter we describe the implementation of the different ECCs in the ScatterWeb OS. The first four paragraphs describe how the ECC library works, how the library is embedded into ScatterWeb OS, which other modules are affected. The subsequent sections deal with the adaptive FEC schemes. The last section describes challenges and problems encountered during the implementation.

## 4.1 libECC

The implementation of the ECC library contains the listed ECCs. The parameters supplied in brackets define the particular ECCs we implemented.

- BCH(63,57)
- BCH(63,51)
- BCH(63,45)
- BCH(63,39)
- BCH(63,36)
- DECTED(16,8)
- Hamming(7,4)
- REP(3,8)

Figure 4.1 gives an overview of the generated code words of these ECCs. It also indicates also the correction power of the ECCs. Moreover, the amount of parity information that is needed to achieve this power is depicted. It is notable that not only the amount of parity information is responsible for the correction power of the ECCs. Consider the example of BCH(63,36), which uses 42.86% of the code word length for parity information and corrects up to five bit errors. Similar relations between the amount of parity bits and the code word size holds for

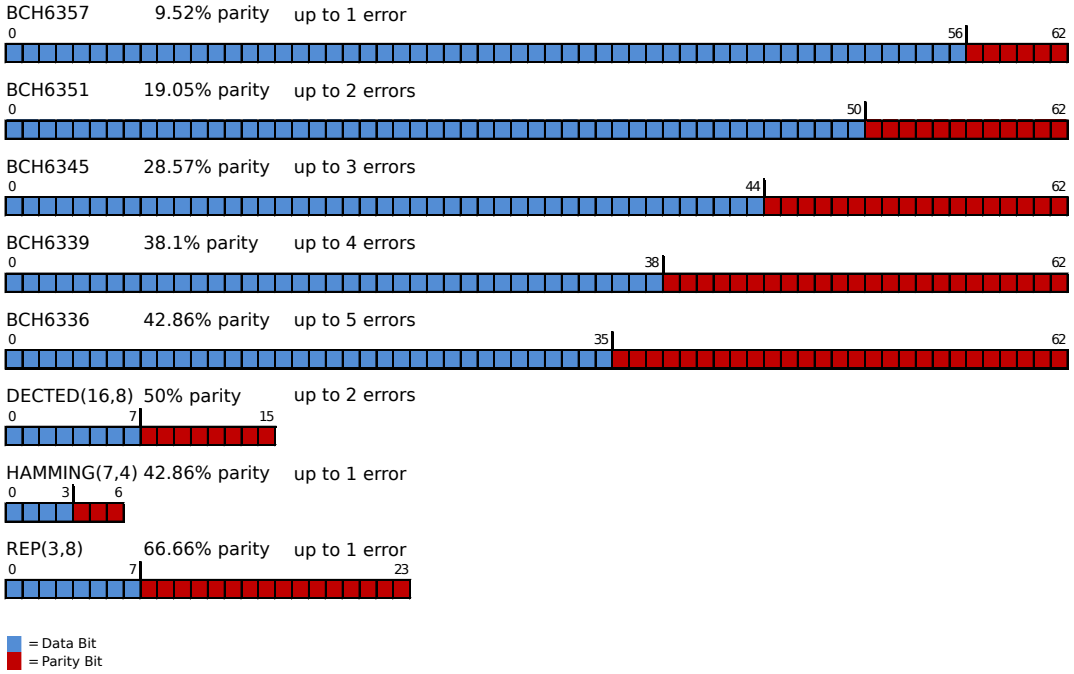



**Figure 4.1:** Scheme of Code Words

the Hamming(7,4) code, although the Hamming(7,4) ECC only corrects up to one error per code word. Generally speaking, the comparison of the code words within a single kind of ECCs shows that the more parity information is appended, the more errors can be corrected. Moreover, the relation between the correction power and the amount of parity information is not linear.

The starting point for the ECC library (ECC module in Figure 3.5) was a small module named libEMPTY. It is a framework for developers to implement their own functionality provided by the developers of ScatterWeb OS. The file ECC.c contains the functionality to encode, correct, and decode a given payload. The file ECC.internals.h contains the definitions of the variables and methods used only within ECC.c. In contrast, the file ECC.h builds the interface to the other ScatterWeb OS modules. The methods and variables there are accessible for other modules. The module is designed in such a way, that most tasks can be done without using extra buffers where the payload to handle is copied to.

We tried to limit the memory footprint of the ECC library as much as possible. The most memory consuming parts of the modules are the matrices, syndrome value lookup tables and polynomials used by the ECCs. This was quite a challenge on the MSB430 nodes, especially because of the constraints with regard to RAM and the runtime stack size. Currently, our implementation supports up to 50 bytes of non-encoded payload data. This limitation comes from the fact that the size of encoded payload grows up to three times the size of the non-encoded payload. We only encode payload and not the header of a packet, in order to keep the impact on the MAC layer limited and the reusability of the library on a higher level.

Our ECC library provides a data structure where the information about the correction and decoding process is stored. This data is collected per received ECC packet (see Section 4.1.3). Such a log entry provides among other information, the number of corrected errors, the size of the decoded payload or the duration of the correction and decoding process.

Another feature of our ECC library is the support for persistent parameter configurations. We use a certain part of the persistent memory of the MSB430 nodes in order to store parameter values. This configuration is restored after shutting down and rebooting a node. Such parameters are for example the ECC, the payload size, the adaptive FEC scheme used, etc. The parameters can be set through commands during runtime.

The implementation of the ECCs is based only on the data types that the mspgcc compiler [31] supports. The library is hence easily portable to other MSP430 sensor platforms such as TelosB [32], Kmote-B [33], etc.

### 4.1.1 Interface to libECC

The ECC library is situated between the Net.c library and the MAC layer. It provides an interface to set up all the relevant parameters for encoding and decoding. This includes configuration routines to configure which ECC is used to encode and decode the payload. The most important method is the one that sets the ECC. The only argument it takes, is the number of the ECC that should be used. The method configures all the required parameters that are relevant for the encoding and decoding process.

```
void ECC_setMode(uint8_t mode);
```

In order to encode a payload, the encode method needs four parameters: a pointer (char* in) to the payload to be encoded, the size (uint16_t in_s) of it, a pointer (char* out) to a buffer large enough for the encoded payload and the size (uint16_t out_s) of the output buffer. The return value of the encoding method is true if no problems occurred while encoding.

```
bool ECC_encode(char* in, uint16_t in_s, char* out, uint16_t out_s);
```

Since the buffer for the encoded payload needs to be provided by the user of the ECC library, the user needs to know the size of the encoded data, in order to reserve enough memory for the output. Therefore, the ECC library provides a method, which calculates the size of the output buffer according to the size of the input buffer, depending on which ECC is used. The same method is used in the reverse direction in the decoding process. There, the method computes the size of the decoded payload according to the size of the encoded payload and the used ECC.

```
uint8_t ECC_calcBufferSize(uint8_t size, bool isEncode);
```

The correction process is completely encapsulated in the ECC library. There is no additional information needed in order to correct possible bit errors. The correction process writes its log information to a special data container named ECC error stats, which contains information about the corrected errors. This information is mainly needed by the adaptive FEC mechanism (see Section 4.2) and is accessible from outside the ECC library.

```
struct _ECC_err_stats {
        uint8_t max;            // max corrected error per code word
        uint8_t min;            // min corrected error per code word
        int err_sum;            // sum of corrected error per payload
        int block_counter;      // num of blocks per payload
} ECC_corrected_error_stats;
```

Some of the methods used in the encoding process are also used for decoding. The computation of the length of the decoded information is done by the same method like mentioned above but only in the reverse way. The decoding is invoked by calling the decode method with four arguments. These arguments are the buffer containing the encoded payload (char* in), the size of encoded payload (uint16_t in_s), a pointer to a buffer (char* out) where the decoded payload should be written to, and the precalculated size (uint16_t out_s) of the decoded payload. The decoding method returns the number of bytes of the effectively decoded payload. This corresponds to the precalculated size of the decoded payload.

```
uint8_t ECC_decode(char* in, uint16_t in_s, char* out, uint16_t out_s);
```

The log entry is written according to the result of the decoding process. Since the log entry also contains header information of the packet, the network module has write access to the fields of the log entry.

```
struct ECC_log_entry {
        uint8_t payload_enc_s;          // size of encoded payload
        uint8_t decBuf_s;               // size of buffer which holds payload
        uint8_t payload_dec_s;          // buffer size for decoded payload
        uint8_t sender;                 // sender from the last hop
        uint8_t origin;                 // initial sender
        uint8_t data_s;                 // size of decoded payload
        uint8_t ecc;                    // used ECC
        uint8_t rssi;                   // received signal strength indicator
        time_t dec_time;                // time used to decode payload
        int num_cor_err;                // corrected error per payload
        uint32_t packet_num;            // ECC packet number
        uint8_t _num;                   // packet number
        bool dec_suc;                   // true if decoding worked sucessfully
        bool crc_mismatch;              // true if packet is corrupted (even after correction process)
        uint8_t rcv_txpwr;              // the transmission power used by the sender
        uint8_t hop_count;              // number of hops the ECC packet needed
        uint8_t isRetrans;              // true if ECC packet is a retransmission
        uint8_t type;                   // packet type
};
```

The most important ECC functionalities and its parameters, such as the ECC itself and the call of the encoding and decoding process, can also be invoked by using the corresponding command. Such commands are entered via the command line, and provides an instant user interface to the ECC library. These commands were essential during testing of the ECC library. The following listing contains the most important commands used for testing.

```
COMMAND( eccmode, 0, cmdargs )   { // sets the ECC parameters }
COMMAND( fecscheme, 0, cmdargs ) { // sets the FEC mechanism }
COMMAND( startetp, 0, cmdargs)   { // starts the experiment on the sensor node }
```

In order to control the adaptive FEC mechanisms, there is a method to tell the node which adaptive FEC mechanism shall be used. For all nodes communicating together, the same FEC scheme needs to be configured. This is done using the method below. It takes the adaptive FEC mechanism as argument and sets up the necessary configuration parameters.

```
void ECC_setFECScheme(uint8_t fec_scheme);
```

### 4.1.2 Error Detection with CRC

The ScatterWeb OS in its standard implementation uses 16-bit CRC for error detection. If the CRC checksum calculated over the payload of the packet on the receiver side matches the checksum appended to the packet, the receiver sends an ACK message. Otherwise, the received packet is corrupted and neglected. CRC itself does not provide error correction. In our implementation of the ECCs, CRC is used for error detection too. Since ECCs only have limited capabilities of error detection, we used CRC for the error detection, instead of solely relying on the detection mechanisms of the ECCs. In contrast to an ECC decoder, a 16 bit long CRC (CRC16) can reliably determine whether a payload contains errors or not. The probability of an undetected error of CRC16 amounts to $1/(2^{16}) \approx 0.015‰$. For a detailed analysis of the collision probability of CRC16 see [34]. For example, Hamming Codes only detect errors up to $d_{min} - 1$ errors [15]. In the case of a Hamming(7,4) ECC $d_{min}$ is equal to three (see Section 3.1.3). Therefore, it only detects up to two flipped bit errors per code word. In contrast to ECC, CRC is able to detect burst errors exceeding the detection capabilities of ECCs. If a code word is affected by burst errors, ECCs may easily exceed their error detection capabilities immediately. A possible way to absorb the impact of burst errors avoiding the usage of CRC is to apply code word interleaving. In this thesis we did however neither investigate the impact of burst errors nor apply code word interleaving.

Figure 4.2 illustrates the application of CRC with the encoding and decoding process in a detailed manner over the link. First, the checksum is calculated over the payload and appended to the end of the payload. Then, the actual encoding is done. The decoding process decodes and corrects the payload if necessary and extracts the original payload with the two bytes of the CRC checksum. To check if the correction process has been successful and the decoded information corresponds to the original payload, again the CRC checksum over the decoded payload is computed and compared with the two bytes at the end. If the checksums match, the decoded information equals the original information transmitted by the sender. The application of CRC hence allows to verify the work of the correction process and the decoder, with only few additional computation power needed for the computation of the CRC checksum.

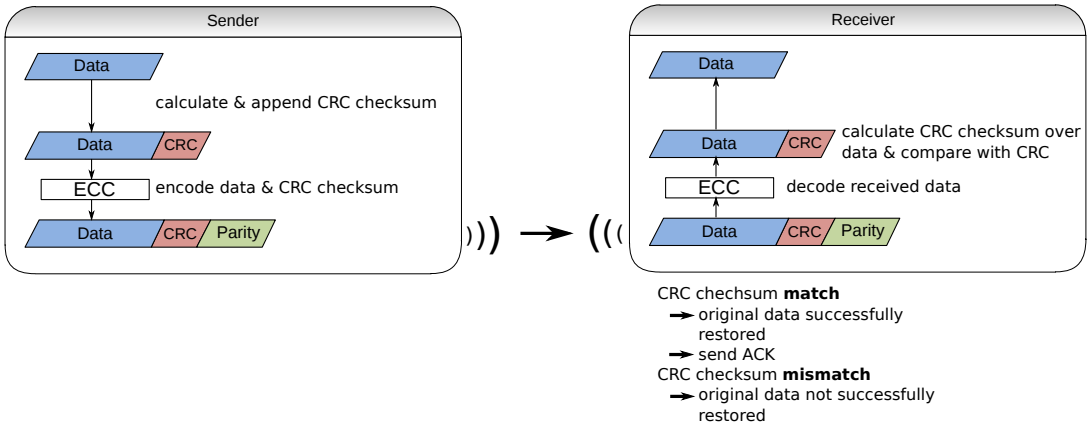



**Figure 4.2:** Error Detection with CRC

### 4.1.3 Example: Transmission and Reception of an ECC Packet

In this paragraph, we briefly illustrate the journey of a packet from the sender to the receiver over a forwarding node. To distinguish a packet not using an ECC-encoded payload from one using ECC, we created a special packet ECC packet labeled ECC_PACKET. It is derived from the USERDEFINED_PACKET which comes with the standard implementation of the ScatterWeb OS. All packets used in ScatterWeb are listed in the file ScatterWeb.Net.PacketTypes.h. The structure of an ECC_PACKET is illustrated in Figure 4.3.

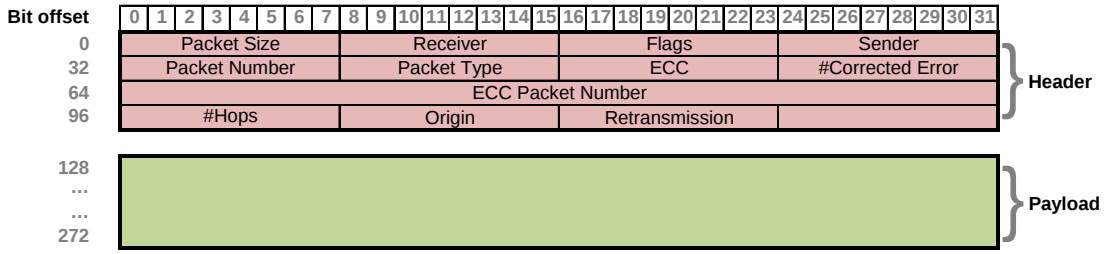



**Figure 4.3:** ECC Packet

The first byte holds the packet size field denoting the size of the entire packet including the header. The following receiver field is a node ID to which the packet should be sent. The flags field contains information about what the packet carries. It indicates for example if the sender of the packet requires an acknowledgement message for the transmission. Moreover, it tells the receiver if the payload of the packet contains a CRC checksum. The sender field contains the ID of the sender node. The sender does not necessarily have to be the same node as the node ID in the origin field. In case of a transmission over multiple hops, the sender could be different from the origin. The packet type field tells the receiver what kind of packet it receives, for example an ECC_PACKET. Besides the packet number which is used by the ScatterWeb OS to match acknowledgements, we introduced the ECC packet number. This field is used in our experiments to track more packets, since the field is 4 bytes long. The packet number which is only 1 byte long can carry a maximum value of 255 and then restarts from 0. The field #Hops counts the hops which an ECC packet took until it reaches its receiver. The retransmission field indicates if the current packet is a retransmission or not.

The example used in Figure 4.4 to illustrate the journey of an ECC packet consists of a topology with three nodes. There is a sender, a receiver and a forwarder between them. The forwarder takes all the ECC packets and sends them to the receiver. We assume that an application in the sender wants to send a certain amount of data with protection through an ECC to the forwarder. The application prepares the data for the network module. The network module then appends the CRC checksum and passes both of it as a payload to the encoder in the ECC module. The task of the encoder is to calculate the parity bits over the payload. For simplification reasons, the figure shows the parity bits at the end of the payload. In reality, the parity is appended after each block if a systematic ECC is applied. After the encoder completed its task, the payload is ready to be placed in an ECC packet. The network module sets up the header for the payload and writes the packet to a buffer. This buffer is repeatedly checked by the ScatterWeb OS, if

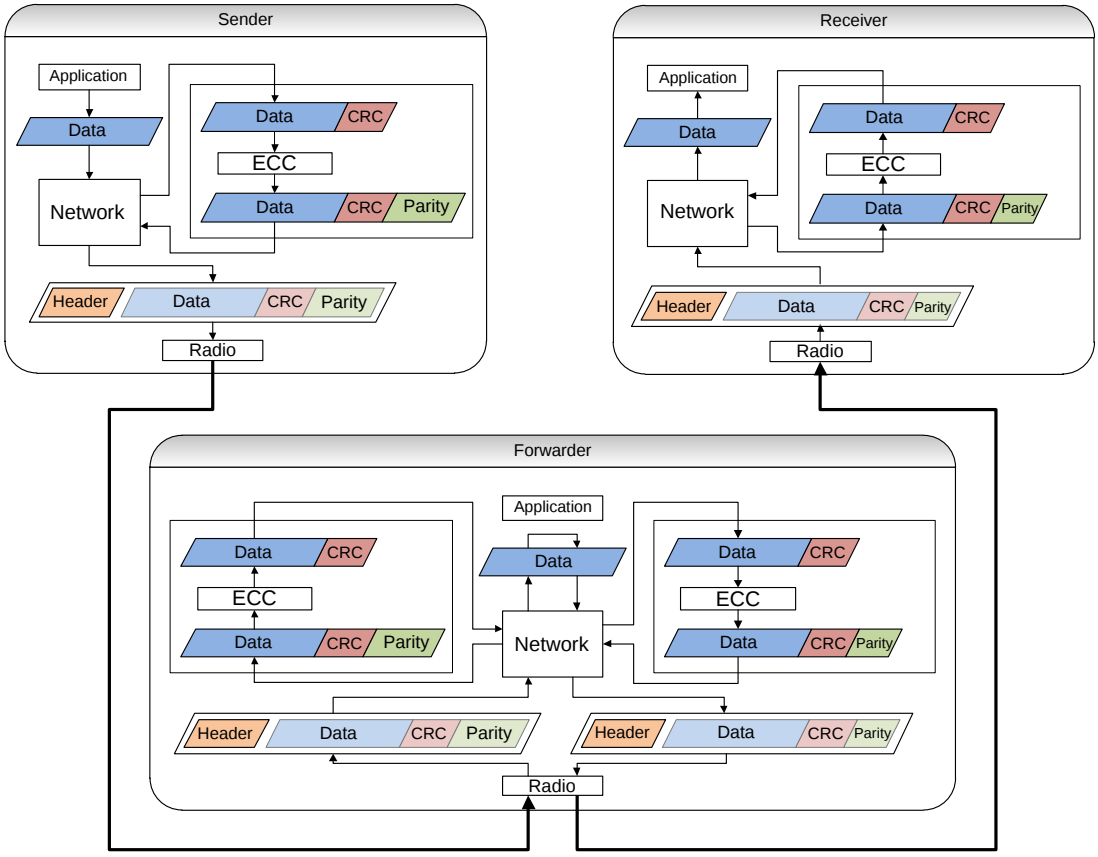



**Figure 4.4:** Journey of an ECC Packet

there are any packets to be sent. If this is the case, it triggers the radio for transmission. After checking if the carrier medium is free, the packet is transmitted over the radio. The ScatterWeb OS uses a simple 802.11-like CSMA on the MAC layer, which was not altered throughout this thesis.

Since every node listens to the medium for a transmission for itself, the forwarder notices that it needs to receive this packet. The radio sends an interrupt signal to the ScatterWeb OS that there is a packet to be received and handled. This forces the network module to take the packet and read the packet type header field. Now, the network module knows what kind of packet it has received and what to do with it. Since the received packet is an ECC packet, its payload needs to be decoded first. The ECC header field indicates which ECC has been used to encode the payload. The network module accordingly sets up the decoder, which takes the payload and starts the decoding process. If it discovers correctable bit errors, the decoder corrects them. The decoder terminates its task by chopping off the parity bits after each block and restoring the blocks to the original data with the appended CRC checksum. Now, the data is passed back to the network module which checks the work done by the decoder. It calculates the CRC checksum

over the data and compares it with the appended CRC bytes. If it matches, the decoder has retrieved the original data.

Since the configuration indicates that the node is a forwarder, the node needs to forward the ECC packet. Considering the configuration, the forwarder knows to which node it has to forward the ECC packet. From this point on, the process of sending and receiving is repeated.

## 4.2 Adaptive FEC Mechanisms in ScatterWeb$^2$ OS

The example described above only uses static ECC configurations. This means that for each hop an ECC packet travels, the applied ECC used to encode and decode is is predetermined. Each sender knows by reading the ECC configuration parameters, which ECC to take for the encoding of any ECC packet that it sends. Right at this point, our approach of dynamically selecting the ECC in a hop-by-hop manner comes into play. Since the link quality is not constant and can vary due to wireless phenomenas, e.g. fading or interferences, we try to optimize the usage of the implemented ECCs depending on the link quality. Since the adaptivity is based on a hop-by-hop manner, our approach adapts to local link quality changes independently from other links in a WSN. The general idea is that the sender checks for each transmission of a packet which ECC would be the most appropriate to take. The optimal selection of an ECC would be the one which adds the less parity as possible but provides enough error correction power to overcome possible bit errors caused by the bad link quality (see Section 3.1). The selection we used for this approach contains the following ECCs in this order of their correction power: OFF (0), Hamming(7,4) (1), DECTED(16,8) (2), BCH(63,45) (3), BCH(63,39) (4) and BCH(63,36) (5) (see Figure 4.5). The numbers on top of the ellipses represent the order of the ECCs. The codes are chosen based on their correction power and the duration of encoding and decoding. Moreover, we wanted to have a representative of each code in the selection. We call our approach Adaptive FEC (A-FEC).

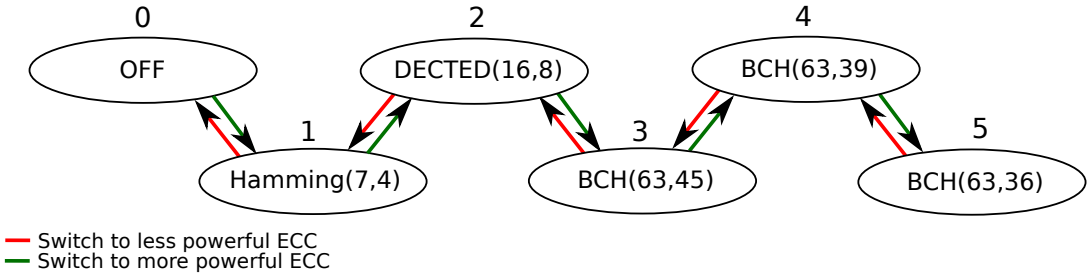



**Figure 4.5:** Correction Power Order of ECCs in A-FEC

In this paragraph, we go deeper into the A-FEC we designed. To be precise, A-FEC consists of three mechanisms. These mechanisms will be explained in the following subsections. In general, they have in common that a sender using one of these mechanisms switches to another ECC depending on knowledge about current and past transmissions. As shown in Figure 2.2, the receiver returns a feedback encapsulated in the ACK message to the sender. The ACK messages are simple header frames with an ACK bit set. This header information is not protected

by ECC, since every node that overhears to a transmission first needs to decode the header to check if the packet is determined for itself, resulting in unnecessary overhead and energy waste. The feedback contains the information about the used ECC for decoding the packet. The ECC returned obviously should be the same as the sender applied for encoding. Additionally, the receiver also adds the maximum number of errors it corrected per code word. For example, if the receiver receives a payload consisting of two code words where the first code word has two flipped bits and the second code word is error free, the maximum number of corrected errors per code word reported in the feedback is two. Unfortunately, it happens that the ACK does not reach the sender of the ECC packet. Like in other reliable communications, the lack of an ACK reception triggers a retransmission of the ECC packet by the sender. A retransmission of an ECC packet in the A-FEC approach is automatically encoded with the next more powerful ECC than the original ECC packet. This reaction is based on the assumption that the original transmission of the ECC packet was not successful due to severe packet corruption, that exceeds the correction power of the ECC used in the first transmission. But what happens when the retransmission could not be delivered or restored too? Then, the receiver never gets a feedback from the receiver. In this case, the different variants of A-FEC approaches are forced to switch to even more powerful ECCs.

As already mentioned in Section 3.2, there have been only few studies about adaptive FEC schemes (e.g. [26], [24]). These approaches were only evaluated in network simulators based on simplistic and user-defined error models. Our A-FEC mechanism adds the new feature of history-based adaptivity, and is evaluated under real-world conditions.

### 4.2.1 Stateless Adaptive FEC (SA-FEC)

The Stateless Adaptive FEC mechanism (SA-FEC) is the simplest adaptive FEC mechanism we implemented. SA-FEC selects the ECC for the next transmission based on the current one. This means that if the current transmission of an ECC packet is successful, it selects the next less powerful ECC mode. Otherwise, it selects the next more powerful ECC mode. The decision depends only on whether the feedback from the receiver is received or not. The sender does not consider the number of corrected errors, which is, besides the used ECC, also reported in the feedback from the receiver. The sender does not keep track of the link quality state. Therefore, we call it *"stateless"*. SA-FEC is very flexible and it reacts quickly to link quality changes, but it is only able to switch between *"neighboring"* ECCs (see Figure 4.5).

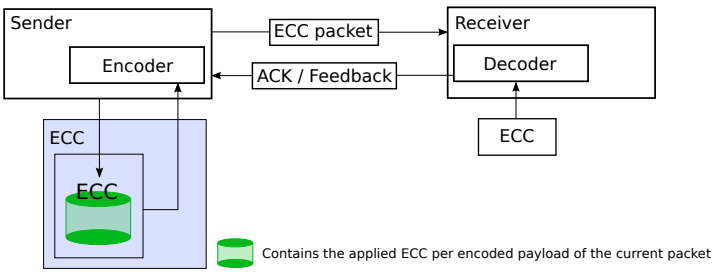



**Figure 4.6:** SA-FEC Scheme

### 4.2.2 Stateful Sender Adaptive FEC (SSA-FEC)

A more sophisticated, history-based adaptive FEC is the Stateful Sender Adaptive FEC mechanism (SSA-FEC). As the name indicates, for each decision it considers a history about past transmissions. In our case, the history contains the last five reported ECCs stored as a ringbuffer. The selection of an appropriate ECC for the next transmission is based on averaging the values stored in the history. Since we are interested in using the "cheapest" ECC, we decrement the next three elements in the ringbuffer if all five values were equal. Obviously, this action has a big influence on the calculation of the average and enforces the selection of the next less powerful ECC (see Figure 4.5). The idea for this behavior is that if the last five transmissions were successfully delivered, the link quality may allow the application of a less powerful ECC. By using the average over the ECC of the last five transmissions, SSA-FEC reacts more smoothly to link quality changes than SA-FEC, and avoids oscillations in selecting ECCs. The averaging also provides a mean to cope with long-term interferences, since the history-based A-FEC mechanisms do not immediately fall back to an ECC with less correction power.

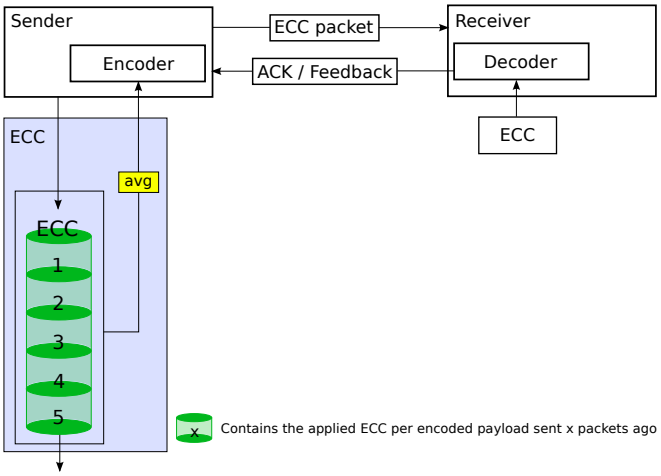



**Figure 4.7:** SSA-FEC Scheme

Figure 4.8 illustrates the behavior of the SSA-FEC approach with an example. Assume a sender already sent four ECC packets using the SSA-FEC approach to a receiver. From the ECC history in the green box we notice that the previous four transmitted ECC packets were encoded with DECTED(16,8) (2). Based on that information the sender computes by averaging that for the transmission of the fifth ECC packet DECTED(16,8) is the most appropriate ECC. The example starts with the transmission of the fifth ECC packet to the receiver. On the receiver side its payload is decoded successfully using DECTED(16,8). Therefore, the receiver returns an ACK message containing the feedback information (ECC: 2). The sender stores this information into the history at the blue position. At this time, one notices that the history only consists of the numbers two. While the sender determines the ECC to apply to the sixth ECC packet, it detects that the last five transmissions already were successfully delivered using DECTED(16,8). This invokes the decrement of the next three elements in the history as described in the upper

paragraph. The reason for the decrement of three out of five elements is that this influences the average in such a way that the sender selects the next less powerful ECC. If we only decrement two elements out of five, the sender would not switch the ECC. Because of the influence of this decrement on the average of the ECC history, the sixth ECC packet is encoded with Hamming(7,4) (1). Since during the transmission of the sixth packet interferences cause severe packet corruption beyond the error correction power of Hamming(7,4), the receiver does not acknowledge the reception to the sender. Consequently, the sender assumes the transmission to have failed and invokes a retransmission after a certain waiting period. Since in our implementation, retransmissions are encoded with the next more powerful ECC, the sixth ECC packet is encoded with DECTED(16,8). In order to accelerate the switch to the next more powerful ECC, the sender already inserts the number of the ECC of the applied ECC to the retransmission into the history. Since the receiver decodes the retransmission successfully it returns an ACK message with the corresponding feedback. This feedback is stored in the history again. For the seventh ECC packet to transmit, the sender determines the Hamming(7,4) ECC to be the most appropriate. After a successful decoding process on the receiver side, the sender receives the feedback and stores it into the history.

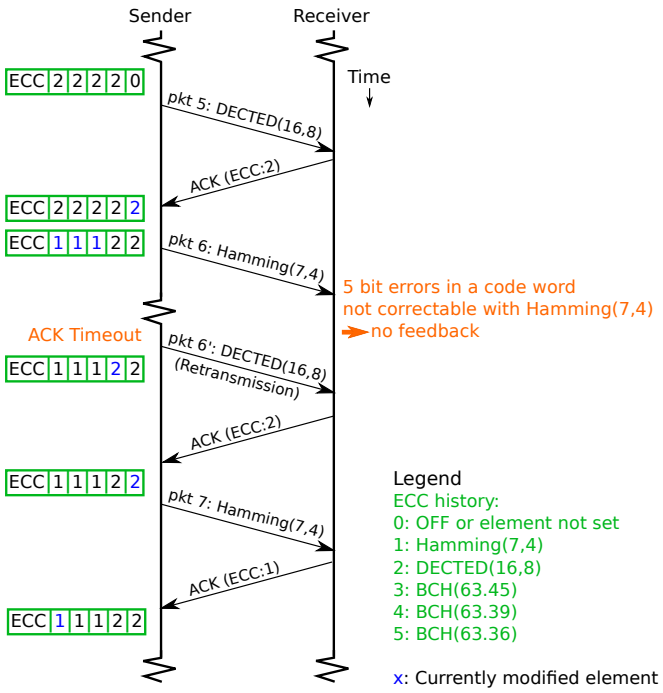



**Figure 4.8:** SSA-FEC Example

### 4.2.3 Stateful Sender Receiver Adaptive FEC (SSRA-FEC)

The Stateful Sender Receiver Adaptive FEC (SSRA-FEC) extends SSA-FEC by using an additional history for the maximum number of corrected errors per code word for each ECC packet. This mechanism uses the full information provided through the feedback of the receiver. Like in SSA-FEC, the sender stores a history of the last five ECCs used. Additionally, the last five numbers of maximum corrected errors per code word are stored. This information in provided through the ACK message from the receiver. For example, if the receiver receives a payload consisting of three code words encoded with BCH(63,45). The first code word is error-free, the second contains one flipped bit, while the third code word has three erroneous bits. The receiver corrects all the errors and generates the ACK message. This message tells the sender that the ECC BCH(63,45) has been used to correct the errors, and that the maximum number of corrected errors per code word was three. The sender then stores this information into the according histories. To determine the appropriate ECC to apply to the next transmission, the sender computes the average of the averages over both histories as depicted in Figure 4.9. This two-step averaging leads to equal weights of both histories for the selection of the next ECC to apply. This slows down the reactivity and decreases the danger of oscillations even more than SSA-FEC. In SSRA-FEC, we used the same condition to decrement three elements of the ECC history as in SSA-FEC.

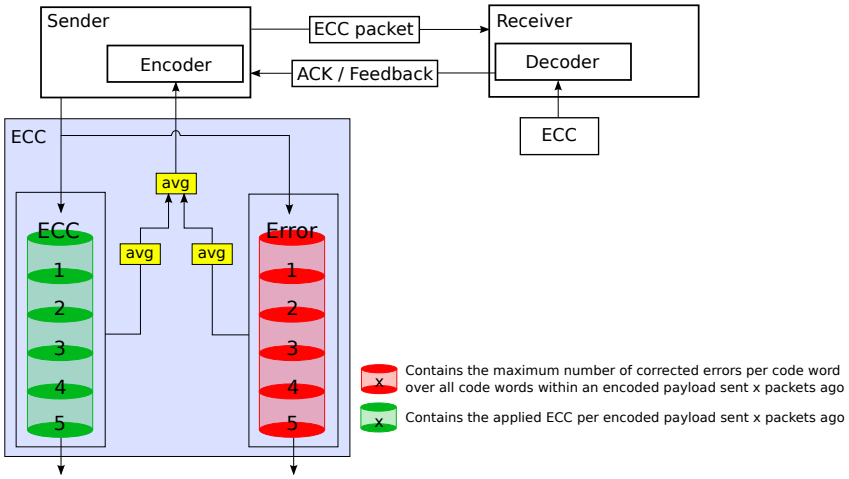



**Figure 4.9:** SSRA-FEC Scheme

As for the SSA-FEC approach the operation of the SSRA-FEC is described through an example. In Figure 4.10 the states of the histories of SSRA-FEC are depicted in the green and red boxes. As in the previous example, the green box represents the ECC history, while the red box contains the number of the maximum corrected errors per code word for every of the last five packets. In this case, we assume that the we already processed four packets. Based on these two histories the sender determines the application of the Hamming(7,4) ECC to packet number five. The receiver decodes the packet successfully using Hamming(7,4), and it detects that every code word in the payload of the ECC packet is error-free. Therefore, the maximum number of corrected errors over all code words in the payload of the ECC packet number five is 0. Subsequently, the receiver generates the feedback out of the decoding information and encapsulates it into the ACK message. Here, the difference between the feedback information of the SSA-FEC approach and the SSRA-FEC approach becomes visible. The feedback in the SSRA-FEC approach contains additionally the maximum corrected error number. This feedback is stored in the corresponding histories as indicated by the blue numbers. The sender determines the next ECC to apply, which is Hamming(7,4) again, and is applied to the sixth ECC packet. Unfortunately, the receiver can not decode the ECC packet correctly, since the error occurred withing a code word exceeds the correction capabilities of Hamming(7,4). Consequently, no feedback message is sent to the sender to confirm the reception. The reaction of the sender to this situation is to invoke a retransmission of the ECC packet. Like for every retransmission in our implementation, the next more powerful ECC is applied to the payload of the retransmitted ECC packet. In this situation the sixth ECC packet contains a payload encoded with DECTED(16,8). To mark the last transmission as failed, the sender stores the ECC applied for the retransmission in the ECC history and inserts a high number of errors into the error history, indicated as the blue numbers in the histories. The reason for this action is the same as in SA-FEC. We are interested in switching to the next more powerful ECC if a transmission failed. Therefore, we insert a high error number to indicate a failed transmission. To continue our example, we observe that the sender receives a feedback from the receiver for the retransmitted ECC packet. The information in the feedback message indicates that the decoding process has been successful and that one bit error was corrected in a code word while all other code words contained no error. This feedback is stored again into the histories like the blue numbers indicate and the next ECC is determined. Obviously, the insertion of the high error number from the failed transmission of ECC packet number six, influences the selection of the ECC for packet number seven. To this ECC packet, DECTED(16,8) is applied.

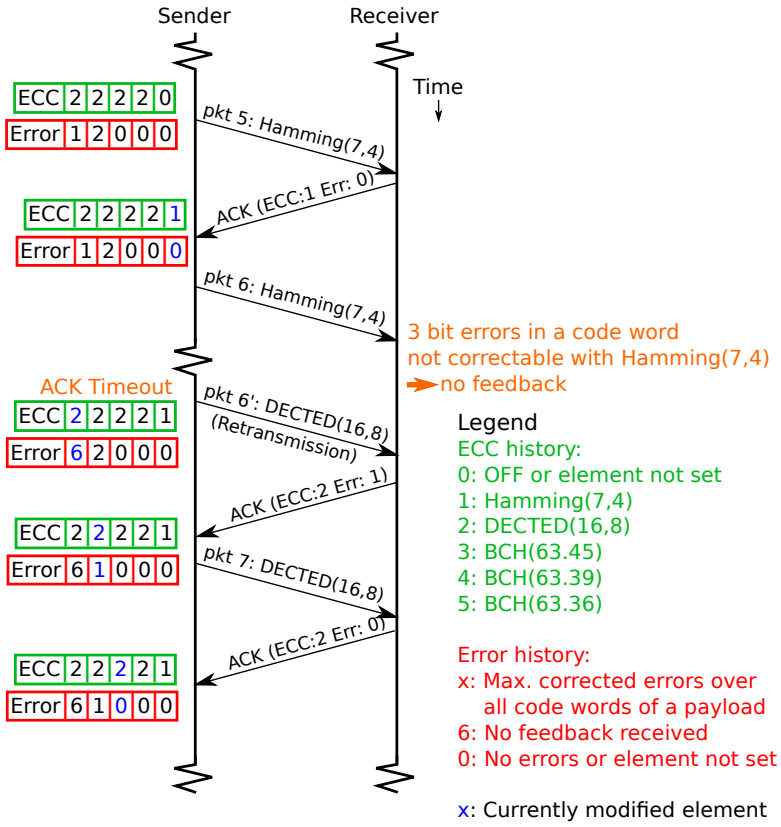



**Figure 4.10:** SSRA-FEC Example

## 4.3 Challenges

We discovered that encoding and decoding can require a non-negligible amount of time depending on the applied ECC. The details will be discussed in Section 5.1. For the implementation of the adaptive FEC mechanisms, we depend on the feedback from the receiver. The ScatterWeb OS standard implementation waits only a few milliseconds for the ACK message before it triggers the retransmission. The default configuration of the waiting time is set to 16 ticks of the radio interrupt, which corresponds to 32 milliseconds. This waiting period initially was too short for the receiver to do the decoding and the correction. Therefore, we needed to extend this waiting period, since otherwise the sender would assume that a packet needs to be retransmitted because of the yet missing ACK message, although the receiver is still in the decoding process and not yet ready to send the ACK. Moreover, the time consumed to perform encoding and decoding varies a lot between the ECCs. This is the reason why we do a dynamic adaptation of the ACK timeout duration, a sender waits for an ACK message depending on the ECC applied to the ECC packet.

Our greatest challenge during the implementation was the small amount of available memory. Some ECCs use lookup tables and matrices to do the computations. The drawback is that

they need quite a lot of memory. Additionally, one has to keep in mind that the implementation of the ECC library is only a part of the network stack. It does not contain any application. Typically, an application would be placed on top of the ScatterWeb OS and needs even more memory. Therefore, the size of libraries should be kept as small as possible in order to provide the free memory space to applications. Since some of the ECCs require quite complex computations to be done, more CPU power would accelerate the encoding, decoding, and correction process. This would allow to decrease the waiting time for the ACK. The fact of the limited resources available restricted the selection of the ECCs to be implemented. The computation power limitation also prevented us from using more than 32 bytes of payload to encode for our experiments.

The selection of the ECCs was also influenced by fact that the MSP430 chip, as any other microcontroller, uses an octet (8 bits) for one byte. In general and in theory, however, the ECCs operate on the bit level, taking blocks of bits which not necessarily correspond to byte boundaries. Therefore, one has to find ECCs that take blocks that fit into one byte on one hand, and on the other hand generate code words that do not wastes bytes. For example, the DECTED(16,8) wastes no bytes since it fits perfectly into the byte boundaries. It takes one byte and generates a code word of two bytes out of it. To find a BCH code that fits more or less to the boundaries was more difficult. For example, the implemented BCH codes use 63 bits for a code word. This means that we loose one bit since each code word needs eight bytes by each block we encode. Not only the length of the code words is a problem but also the block size. Consider the example of encoding four bytes with BCH(63,36). This means there are four unused bits that need to be encoded and sent. Besides this overhead, another problem in the decoding process occurs. Since the decoder only receives encoded data of eight bytes length in case of using BCH codes, it needs to know how much of the decoded data is effectively original data, that the sender encoded. To overcome this problem, we use the fact that for every of the implemented BCH codes, we do not fill all bits with original data in one block. The unused bits are not wasted, but carry information about how many bytes in the block effectively contain original data.

Another challenge were the problems that occurred through concurrency. Since the ScatterWeb OS is interrupt driven and our computations take quite a lot of time, we needed to protect the parts of decoding and encoding from interruption through other events. This includes the access and modification of packet buffers which are modified from two events at the same time. The ScatterWeb OS does not provide this protection by itself and had to be introduced by mutex schemes. Such problems occurred mostly in scenarios where several nodes communicate together and high packet rates were examined.

As described in [35], results from experiments in real-world wireless sensor testbeds are very hard to reproduce. This is a drawback from real-world experiments compared to experiments received from a wireless network simulator. Such simulators allow to do a fine grained configuration of the environmental conditions, which influence the behavior of the sensor nodes. Moreover, this eases the reproduction of results. Since wireless network simulators always relay on a certain link error model, one can questing the accuracy of these modules. We explicitly wanted to do the experiments in a real-world environment, in order to see how the communication within a real-world WSN can benefit from applying ECCs.

Chapter 5

# Experimental Evaluation

In this chapter, we evaluate our ScatterWeb OS ECC implementation with respect to computational complexity and resilience against errors in different experimental scenarios. The first section compares the computational costs of the different ECCs. Subsequently, in Sections 5.3.1, 5.3.2, 5.3.3 and 5.3.4, we evaluate the different ECCs schemes - including the adaptive schemes - in several experimental scenarios ranging from the single-link case to a multi-link network. Finally, we discuss the results of the evaluation of the multi-link scenario.

## 5.1 Computational Performance

For the performance evaluation of the implemented ECCs, we used a single MSB430 sensor node. We measured the time needed for encoding different payload sizes, ranging from 3 to 52 bytes of data and for decoding the corresponding code words. When measuring the decoding duration, bit errors were introduced into the payload from zero to the amount of errors which the ECCs can correct per code word. This means for example that the Hamming(7,4) decoder which can correct up to one error per code word was tested with zero and one error per code word, but not more. For each of these configurations, the encoding and decoding performance was measured during 1000 runs. To measure the encoding and decoding duration we used the internal clock of the MSB430 sensor nodes.

### 5.1.1 Encoding

There is a trade-off between the correction capability of ECCs and their complexity. To support this observation, the time needed for encoding and decoding a certain amount of payload with the different ECCs was measured. The encoding time is the duration a node needed to encode the payload. Therefore, the clock value before calling the encode method is stored as well as the clock value after the encoding. The difference between the two timestamps corresponds to the time the MSB430 needs to process the instructions of the encoding method. When the function call returns from the encoding method, the data is encoded.

```
// measure encoding time
time_t diff_enc;                                    // variable to store the encoding duration
diff_enc.secs = 0;
diff_enc.millis = 0;
time_t now = CURRENT_TIME;                          // stores the current clock value before encoding
ECC_encode(char* in, uint16_t in_s, char* out, uint16_t out_s);
time_t later = CURRENT_TIME;                        // stores the current clock value after encoding
TIME_DIFFERENCE(later, now, diff_enc);              // write time difference to the diff_enc variable
```

Figure 5.1 shows the encoding duration versus the amount of payload bytes to be encoded. The BCH code exhibits a characteristic step-shaped pattern in the encoding time. These steps can be explained by the blockwise encoding process of the BCH code. The smallest block size unit of the BCH code has been set to four bytes. If only one byte has to be encoded, an entire block needs to be allocated, thereby "wasting" three bytes. If the number of bytes to encode exceeds the amount of bytes one block can take, a new block needs to be filled. This effect manifests itself as a lift in the graphs representing the encoding duration of the BCH ECCs and is depicted in Figure 5.1. Moreover, the length of a step depends on the size of the block. For example, the BCH(63,57) code has the largest blocks, resulting in the longest steps. Another observation is that for the BCH codes, the more parity information is added, the longer the encoding takes. The same observation can be made comparing Hamming(7,4) and DECTED(16,8). DECTED(16,8) needs around 7.14% more parity information than Hamming(7,4). But this observation is not valid for all ECCs compared together. The time needed for encoding does also heavily depend on the amount of operations the ECC needs to encode a block. That may also be the reason why the Hamming(7,4) and the DECTED(16,8) need more time to encode than the BCH codes. Both, Hamming(7,4) and DECTED(16,8) need many matrix operations to encode a block, while BCH codes only performs computationally cheap shifts in memory and less computation operations. In general, the encoding duration grows linearly with the amount of bytes to encode. It is also obvious that REP(3,8) is the fasted ECC, since it needs almost no computation and only consists of memory copying operations.

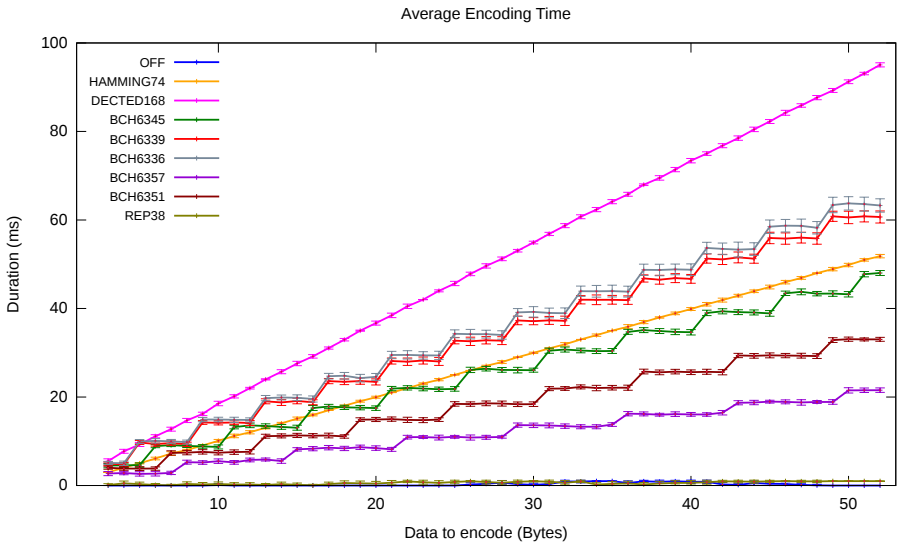



**Figure 5.1:** Average Encoding Time vs. Bytes to Encode

### 5.1.2 Decoding

To measure the decoding time needed we applied the same method like in the encoding time measurement. The timestamp before the call of the decoding procedure is subtracted from the timestamp stored right after the decoding process has finished. The decoding procedure contains the error correction and the recovering of the original information.

```
// measure decoding time
time_t diff_dec;                                    // variable to store the decoding duration
diff_dec.secs = 0;
diff_dec.millis = 0;
time_t now = CURRENT_TIME;                          // stores the current clock value before decoding
ECC_decode(char* in, uint16_t in_s, char* out, uint16_t out_s);
time_t later = CURRENT_TIME;                        // stores the current clock value after decoding
TIME_DIFFERENCE(later, now, diff_dec);              // write time difference to the diff_dec variable
```

Figure 5.2 depicts the decoding durations for each amount of errors per encoded block that can be corrected by the ECCs. All the ECCs displayed in one picture were configured to correct the same amount of bit errors that were distributed randomly over all code words. The number of bytes in the x-axis corresponds to the size of the decoded data. Like this, the decoding durations could be matched to the encoding times. In every diagram, only the ECCs are shown that are able to correct the amount of errors for which the diagram is drawn. Hence, the graph showing the decoding duration for OFF appears only in the diagram where zero errors per code word are plotted.

A general observation is that the more errors occur, the more time is needed to decode. Surprisingly we observed that for the DECTED(16,8) ECC it is not relevant if it needs to correct one error or two errors per code word. In other words, it is almost as fast to correct and decode code words with two errors as with one error. This is explainable by the way errors are found in DECTED(16,8). The implementation of DECTED(16,8) uses lookup tables with error codes. These error codes indicate the error positions independently from the number of errors. Although a lookup of a two bit error is more expensive than a lookup of a single bit error, the difference in the decoding duration proved to be negligible.

For the Hamming(7,4), the DECTED(16,8) and the BCH ECCs one observes that the difference in the time to decode and correct no error to one error is higher than the increment in time for the other number of errors. The explanation for this observation is that all the mentioned ECCs are finished with the correction process if the so-called syndrome values are zero. If the latter is the case, there are no errors to handle (see Figures 3.1, 3.3). If however there are errors, the syndrome has a non-zero value and the errors need to be located and calculated, which obviously takes more time. Moreover, the number of errors and their location within the code word, has no big influence on the decoding duration.

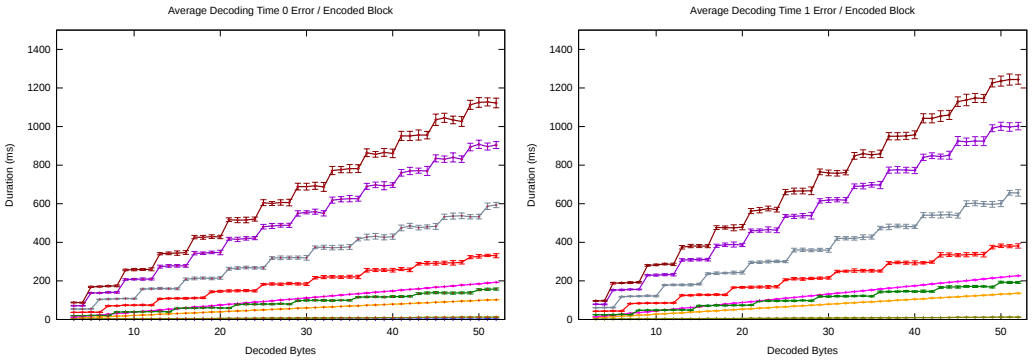
Considering the BCH ECCs, one notices again the steps similar to the steps in the encoding time measurement (see Figure 5.1). These steps are well-explainable with the blockwise encoding of BCH, which similarly impacts on the encoding times.

An interesting fact is that for the decoding time of REP(3,8) it does not matter if there are errors to correct or not. The REP(3,8) always does the same operations no matter how many bits are flipped.

We observed that the BCH ECCs are more time consuming for correcting and decoding compared to the other ECCs. This circumstance comes from the basic design difference of the correction process between BCH codes and the other ECCs. While BCH codes use an iterative

and therefore more expensive algorithm to find and correct errors, the other ECCs use lookup tables or even simpler algorithms to find and correct errors.

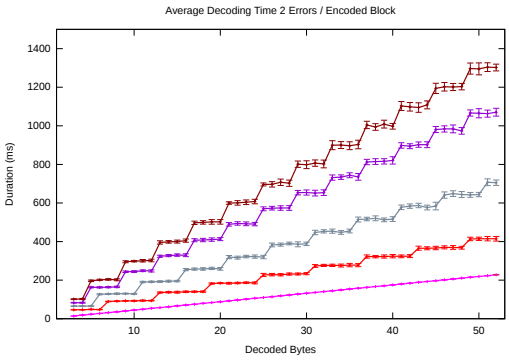
The comparison of the encoding durations with the decoding durations shows that for all ECCs decoding takes much longer than the encoding process. The reason for this is that the data processed by the decoder is larger and more memory copy and comparison operations used to be performed. The x-axis in th Figure 5.2, shows the number of bytes of the decoded data. Since the ECCs operate on the bit level and not on bytes, a large number of byte-to-bit array conversions and vice versa need to be done, which include time consuming data shift and copy operations in the memory.



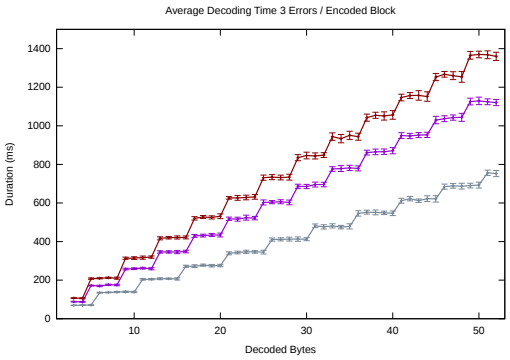

(a) 0 Error / Encoded Block



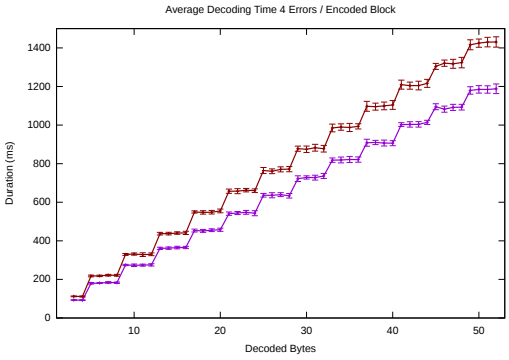

(b) 1 Error / Encoded Block



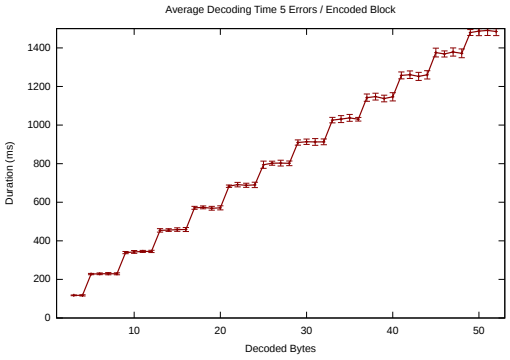

(c) 2 Errors / Encoded Block



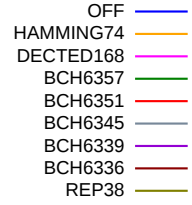

(d) 3 Errors / Encoded Block




(e) 4 Errors / Encoded Block




(f) 5 Errors / Encoded Block




(g) Legend

**Figure 5.2:** Average Decoding Time vs. Decoded Bytes

## 5.2 Energy Cost Estimation of Error Correcting Codes

ScatterWeb OS only alternates between two operation modes of the MSP430 microcontroller chip [36], namely the fully active mode and the Low Power Mode 1 (LPM1). We measured the current draw of the MSB430 sensor nodes with the CPU in both of these modes. In order to better differentiate the current gap between the two measurements, we turned off all the onboard sensors, as well as the CC1020 radio chip.

Figure 5.3 depicts an excerpt from the two traces measured using the Sensor Node Management Device (SNMD) devices from TU Karlsruhe [37]. One can clearly see the difference between the fully active mode where the CPU runs at full speed (11 MHz) and the low power mode LPM1. For example, the CPU is in the active mode when the node encodes or decodes a packet payload. In LPM1, the CPU and MCLK (Master Clock) are disabled, but timers and peripheral interrupts are still enabled, which can wake up the CPU within less than $1\mu s = 10^{-6}$ seconds. The stable average values of the entire node's current draw accounted to 3.75 mA for the fully active operation mode and 1.92 mA for the LPM1. This difference might vary to some few percent dependent on the node instance chosen for the measurement, since different nodes - even if they are of the same type - exhibit small variations in their current draws, as pointed out with the MSB430 platform in [38].

With the measurement of the current difference depicted in Figure 5.3, we can derive a cost function for the encoding and decoding function, taking into account the time it takes to encode or decode a payload and the higher power consumption in the active CPU mode.

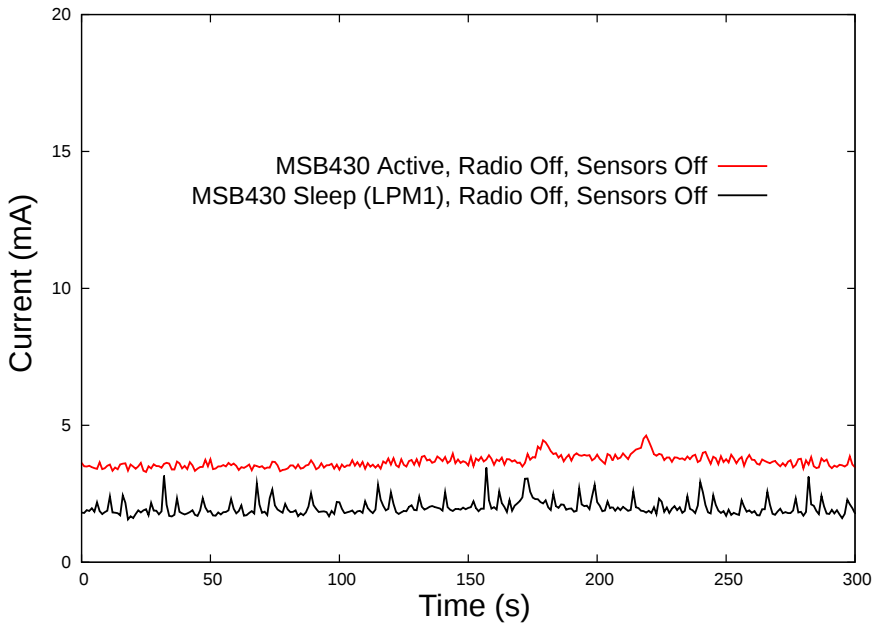



**Figure 5.3:** Power Trace of the MSB430 Sensor Node

The power consumption of a node $P_S(t)$ follows the power relationship

$$P(t) = V(t) \cdot I(t)$$

We define $P_{Default}(t)$ as the power consumption function of the node without using FEC, and $P_{FEC}(t)$ as the power consumption function of the node using FEC.

We define the *cost* of the application of FEC as the additional power consumed by the node when encoding and decoding. The power cost function $P_{cost}(t)$ can then be denoted as

$$P_{cost}(t) = P_{FEC}(t) - P_{Default}(t)$$

Integrating the above measured values into the equation yields:

$$P_{cost}(t) = (I_{FEC} - I_{Default}) \cdot U_{supply} = 1.83mA \cdot 4V = 7.3mW$$

$I_{FEC}$ corresponds to the average current used when the CPU is in the active mode, while $I_{Default}$ is the average current used in LPM1. We briefly give an example to illustrate the application of the derived cost function: The energy cost $E_{enc/dec}$ of an encoding or decoding operation which takes $200ms$ calculates as

$$E_{enc/dec} = 200ms \cdot 7.3mW = 1.46mJ$$

The encoding / decoding durations from Sections 5.1.1 and 5.1.2 can hence be linearly mapped using the cost function $P_{cost}(t)$ to energy cost functions using the measured power gap between the active CPU state and the LPM1.

## 5.3 Error Correction Performance

In this section we present the results of the error correction performance of the ECCs in different topologies. The following paragraphs evaluate the single-link scenarios, while the last paragraphs discuss the evaluation of the multi-link topology.

For the better understanding of the used power settings, Figure 5.4 depicts the output power of the CC1020 transceiver as specified in the manufacturer's datasheet in [4] . On the x-axis. the adjustable power ticks are listed. The CC1020 can be set by software to operate with a certain transmission power. Since the values on the x-axis do not have any unit, we call them ticks.

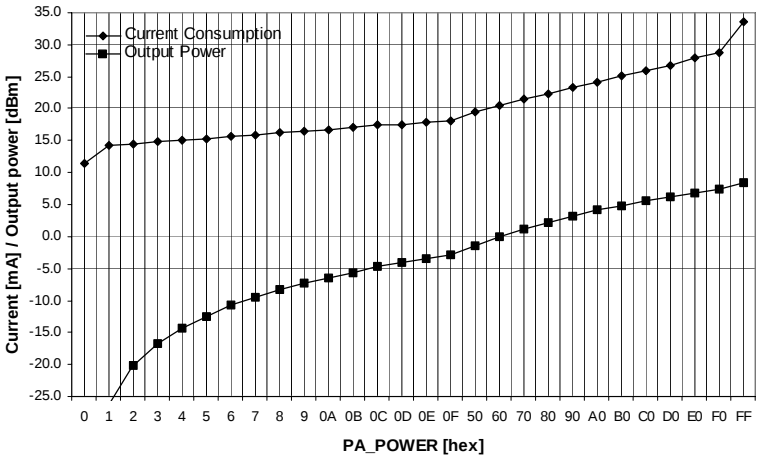



**Figure 5.4:** CC1020 Output Power [4]

For all the experiments, we used the default CSMA MAC protocol integrated into the ScatterWeb OS, which applies an 802.11-like carrier sensing and random back-off mechanism. Contention-based MAC protocols such as X-MAC [39] or B-MAC [40] are predominantly applied in most of today's deployment studies. Since carrier sensing operations are inherently unreliable on low-power radio transceivers such as the CC1020, collisions are a frequent reason for transmission failures. We examine the performance of static and adaptive FEC under concurrent traffic in the multi-hop scenario in Sections 5.3.3 and 5.3.4.

For each link (indoor and outdoor) the same settings considering number of transmitted packets, transmission powers as well as the time interval between two ECC packets transmissions were used. For each link the data of one run is evaluated.

### 5.3.1 Indoor Single-Link Scenario

In the following paragraphs, we discuss the results of indoor single-link measurements. The link was set up between two MSB430 sensor nodes (msb1 and msb3) on channel 3 inside the building of the Institut für Informatik und angewandte Mathematik, Neubrückstrasse 10, 3012 Bern (IAM). The sensor node msb1 was placed in room 314 on the third floor of the IAM building. The other sensor node, msb3, resided in room 304 on the same floor. The link is depicted in

Figure 5.5, where msb3 was the sender and msb1 was the receiver. For every static ECC and every A-FEC mechanisms, a single run consisting of 1000 sent packets per transmission power value was measured. Every run took at least 2000 seconds. We tested all the implemented ECCs. We evaluated 15 transmission power settings ranging from 1 tick (≈ -25 dBm) to 15 ticks (≈ -3.5 dBm). Every two seconds, a packet was transmitted. For each ECC packet, 32 bytes (plus 2 bytes CRC) of payload were encoded. In total, this setting leads to 180000 transmitted packets with a total experiment duration of 100 hours. For the static ECC measurements we iterated over every ECC and changed it for each transmission, in order to achieve comparable conditions of the channel for all ECCs. Additionally, the PDR of all three A-FEC mechanisms were measured. Since in the case of the static ECC measurements no retransmissions were sent, for the A-FEC mechanisms we only consider the transmission of the original ECC packet. This creates comparable conditions for all of the static ECCs and the A-FEC mechanisms. The measurements were captured during the night and on weekends in order to minimize environmental interferences.

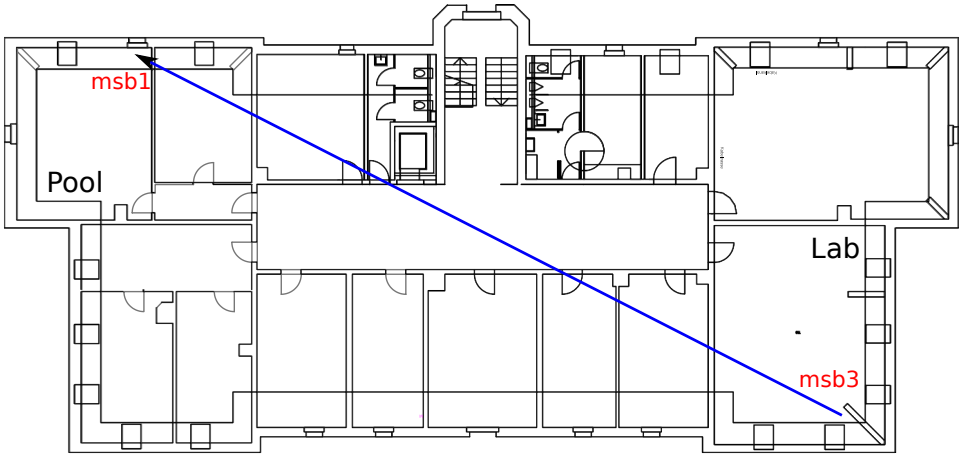



**Figure 5.5:** Indoor Link

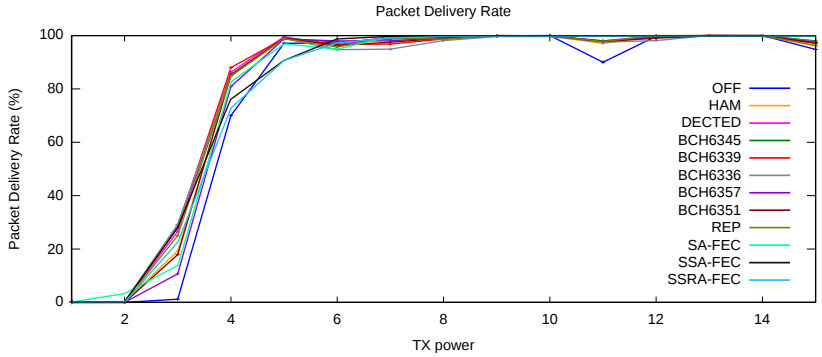
### Packet Delivery Rate (PDR)

Figure 5.6(a) depicts the PDR for each ECC versus the used transmission power setting. The general observation is that in most cases the PDR is increased using ECCs to protect the payload of packets against bit errors. Between TX power 5 ticks (≈ -12.5 dBm) to 10 ticks (≈ -6.5 dBm), the usage of ECCs leads to a slightly lower PDR than using no ECC. Since in the implementation only the payload is protected against bit errors, bit errors in the packet header can decrease the PDR. Moreover, the length of the packets varies depending on which ECC is being used. Therefore, using no ECC results in the smallest packets, which are less affected by bit errors.

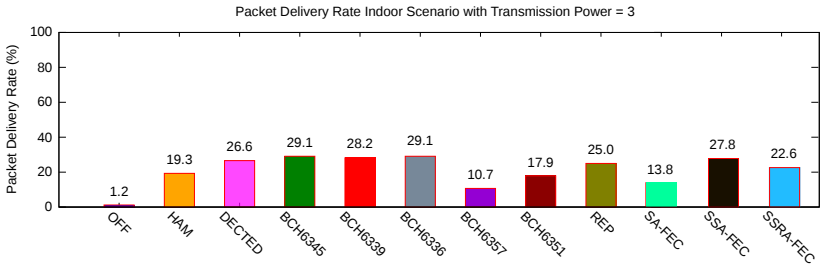
If the transmission power is above 5 ticks (≥ -12.5 dBm), the benefits of ECCs are very small. Since an increased transmission power leads to a higher Signal-to-Noise Ratio (SNR), less errors occur. The less errors occur, the lower the benefit of using ECCs. The application of ECCs can primarily show its potential in situations where errors occur. Figure 5.6(b) is a

detailed analysis when setting the transmission power to 3 ticks (≈ -17 dBm). In this case, no matter which ECC is applied, the PDR is increased. One can see that the ECCs, which correct more than two errors do not increase the PDR in a significant way compared to the ECCs which correct up to two errors. This could indicate that most corrected errors are one or two bit errors.

The A-FEC mechanisms achieved similar results as the static ECC PDR measurements. For every A-FEC approach and every transmission power value, we measured 1000 packets, where every two seconds a packet was sent. Considering low transmission powers, especially SSA-FEC performs well. It achieves not the best results but is close to the powerful BCH ECCs which achieve slightly better results.




(a) PDR




(b) PDR with TX Power 3 Ticks (≈ -17 dBm)

**Figure 5.6:** Packet Delivery Rate Indoor Link

Since the A-FEC mechanisms select the ECCs depending on the link quality, it is interesting to investigate the PDR distribution over the selected ECCs. The following Figures 5.7(a), 5.7(b) and 5.7(c) illustrate the breakdown of the PDR values for every transmission power and ECC setting. The experimental data for these plots are the same as used to illustrate the PDR values in the previous paragraph. For the readability the percentages for every ECC have been left out. The percentages can be found in the Tables 5.1, 5.2 and 5.3. One clearly observes that for low transmission power values only powerful ECCs such as BCH(63,39) and BCH(63,36) could deliver the ECC packets successfully. Therefore, the highest PDR values were achieved by these ECCs. Using transmission power higher than 4 ticks (≥ -15 dBm) most ECC packets were successfully delivered without being encoded by an ECC. This is indicated by the red parts

of the columns. Since the SNR increases using higher transmission power values, the less errors occur and the less powerful ECCs are chosen. Although sometimes still errors occurred, the application of ECCs with a low correction power were used such as Hamming(7,4) (green part). This also shows the advantage of applying an adaptive FEC mechanism. Naturally, high PDR values were also achieved with powerful ECCs, but they are expensive in terms of energy costs. Therefore, it is more appropriate to apply an A-FEC mechanisms which applies the powerful and costly ECCs only when they are needed.

| TX Power | OFF | HAM | DECTED | BCH6345 | BCH6339 | BCH6336 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0.5 | 1.0 | 0.7 | 0.7 | 0.2 | 1.0 |
| 3 | 0.1 | 0.2 | 0.2 | 1.2 | 4.2 | 7.7 |
| 4 | 71.6 | 6.8 | 2 | 2 | 0.3 | 0.1 |
| 5 | 92.8 | 4.1 | 0.1 | 0 | 0 | 0 |
| 6 | 88.2 | 5.9 | 0.8 | 0.1 | 0 | 0 |
| 7 | 96.5 | 3.3 | 0 | 0 | 0 | 0 |
| 8 | 95.4 | 4.6 | 0 | 0 | 0 | 0 |
| 9 | 96.1 | 3.9 | 0 | 0 | 0 | 0 |
| 10 | 94.4 | 5.6 | 0 | 0 | 0 | 0 |
| 11 | 96.3 | 3.7 | 0 | 0 | 0 | 0 |
| 12 | 95.3 | 4.7 | 0 | 0 | 0 | 0 |
| 13 | 96.8 | 3.2 | 0 | 0 | 0 | 0 |
| 14 | 97.9 | 2.1 | 0 | 0 | 0 | 0 |
| 15 | 97.5 | 2.5 | 0 | 0 | 0 | 0 |

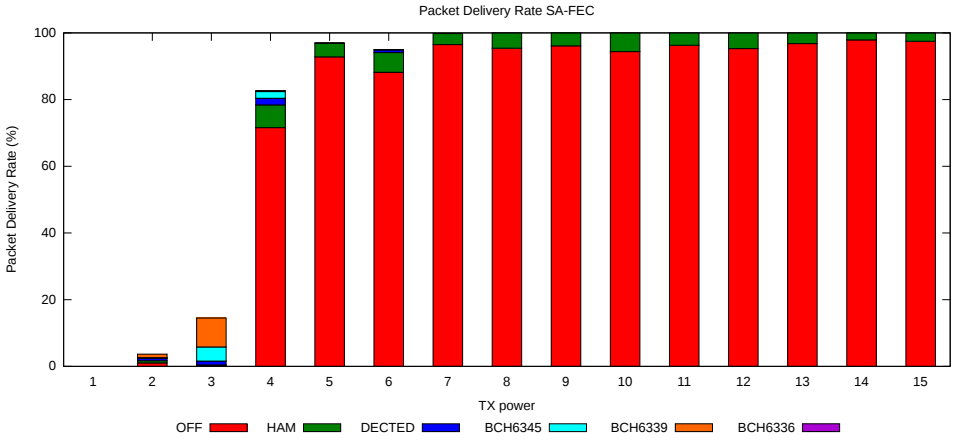
**Table 5.1:** PDR Distribution Indoor Link SA-FEC

| TX Power | OFF | HAM | DECTED | BCH6345 | BCH6339 | BCH6336 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0.1 | 0.2 | 8.3 | 19.2 |
| 4 | 33.1 | 35.2 | 7.1 | 0.7 | 0 | 0 |
| 5 | 71.3 | 17.9 | 1.5 | 0 | 0 | 0 |
| 6 | 91.9 | 6.7 | 0.2 | 0 | 0 | 0 |
| 7 | 93.7 | 6 | 0.1 | 0 | 0 | 0 |
| 8 | 93.7 | 6.3 | 0 | 0 | 0 | 0 |
| 9 | 95.8 | 4.1 | 0.1 | 0 | 0 | 0 |
| 10 | 95.3 | 4.6 | 0.1 | 0 | 0 | 0 |
| 11 | 94.4 | 5.5 | 0.1 | 0 | 0 | 0 |
| 12 | 96.1 | 3.9 | 0 | 0 | 0 | 0 |
| 13 | 96.3 | 3.7 | 0 | 0 | 0 | 0 |
| 14 | 95.4 | 4.5 | 0.1 | 0 | 0 | 0 |
| 15 | 95.8 | 4.2 | 0 | 0 | 0 | 0 |

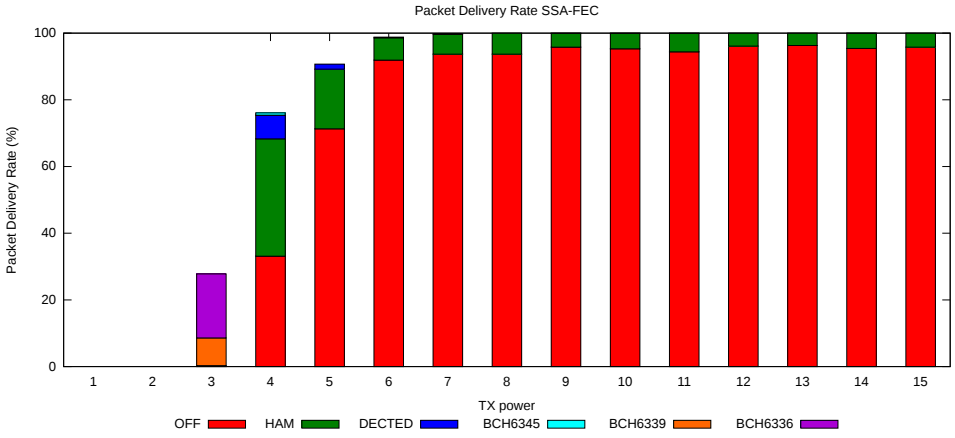
**Table 5.2:** PDR Distribution Indoor Link SSA-FEC

| TX Power | OFF | HAM | DECTED | BCH6345 | BCH6339 | BCH6336 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1.1 | 8.5 | 13 |
| 4 | 30.1 | 37.8 | 5 | 0.1 | 0 | 0 |
| 5 | 89.6 | 1.2 | 0 | 0 | 0 | 0 |
| 6 | 97.4 | 0 | 0 | 0 | 0 | 0 |
| 7 | 98.7 | 0.5 | 0 | 0 | 0 | 0 |
| 8 | 99.8 | 0 | 0 | 0 | 0 | 0 |
| 9 | 99.9 | 0 | 0 | 0 | 0 | 0 |
| 10 | 99.9 | 0 | 0 | 0 | 0 | 0 |
| 11 | 99.9 | 0 | 0 | 0 | 0 | 0 |
| 12 | 100 | 0 | 0 | 0 | 0 | 0 |
| 13 | 100 | 0 | 0 | 0 | 0 | 0 |
| 14 | 100 | 0 | 0 | 0 | 0 | 0 |
| 15 | 100 | 0 | 0 | 0 | 0 | 0 |

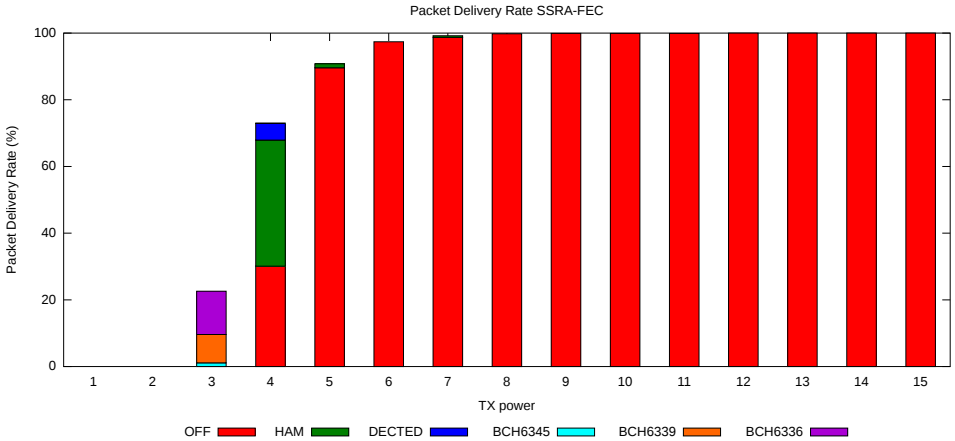
**Table 5.3:** PDR Distribution Indoor Link SSRA-FEC




(a) PDR SA-FEC




(b) PDR SSA-FEC




(c) PDR SSRA-FEC

**Figure 5.7:** Packet Delivery Rate Distribution Indoor Link A-FEC

### ECC Selection Behavior

In this paragraph, we discuss the behavior of the A-FEC mechanisms on the indoor link. The results originates from the same data that are used for the evaluations in the previous paragraphs. The Figures 5.8, 5.9 and 5.10 depict for every A-FEC mechanism depending on the applied transmission power which ECCs were selected for how many sent ECC packets. The y-axis representing the number of packets is in logarithmic scale. For every ECC there is a colored column which indicates for how many ECC packets a particular ECC has been applied. As known from the experiment configuration, for every transmission power setting 1000 ECC packets were sent. Therefore, the sum off all columns of a particular transmission power setting is equal to 1000.

The general observation is that for low transmission power settings every A-FEC mechanism applied a powerful ECC such as BCH(63,36) (violet columns). For transmission power settings higher than 3 ticks ($\approx$ -17 dBm) mostly no ECC was used. In the region from 3 to 4 ticks the A-FEC mechanisms exhausted the bandwidth of the ECC selection at most. A explanation for this situation is that these transmission power settings were high enough for the receiver to detect that a packet was sent to it, but too low to reach a SNR level, where interferences only had a very limited influence.

Although for high transmission power settings most ECC packets were not encoded (red columns), for a few ECC packets still an ECC with low correction capabilities was selected (see Figures 5.8, 5.9). A reason for this behavior could be that short-lived interferences caused by environmental conditions affecting only few continuous packets caused SA-FEC and SSA-FEC to quickly switch to Hamming(7,4) (green columns).

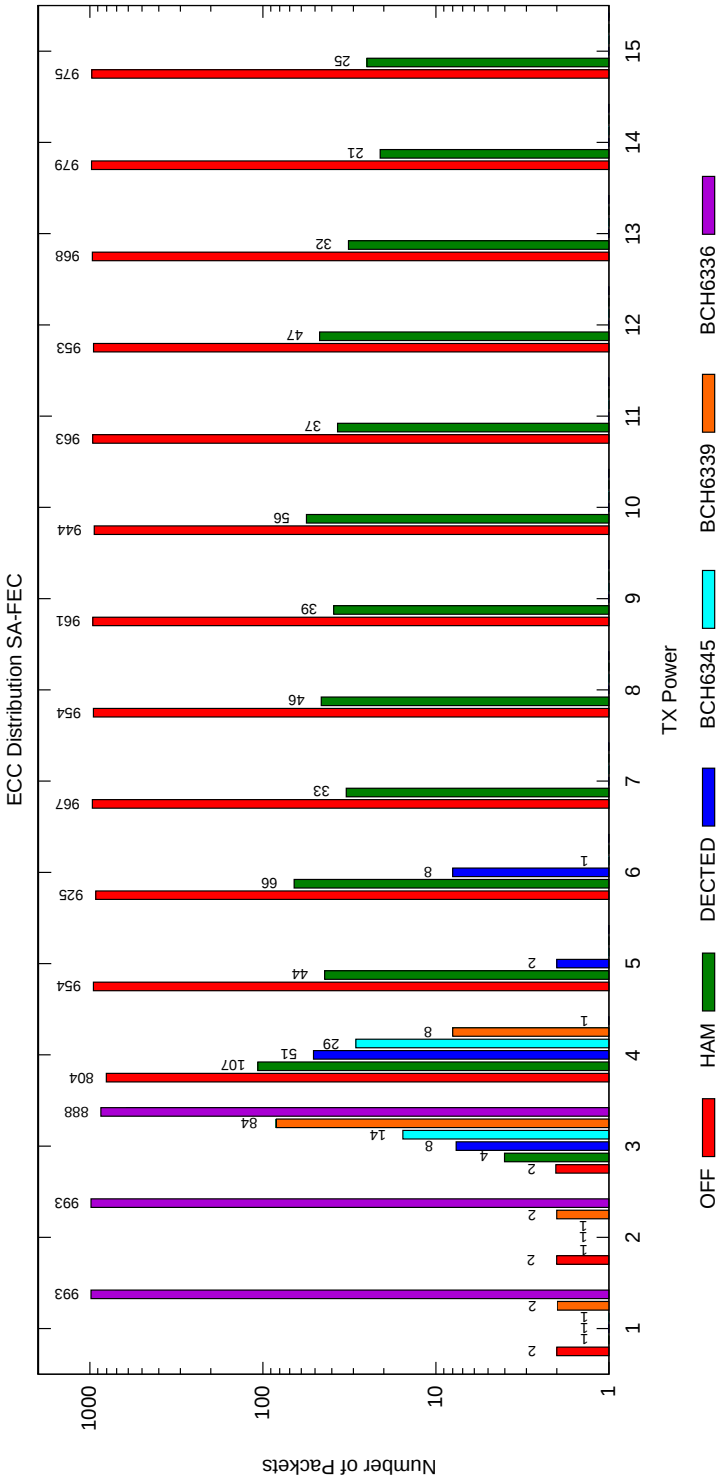



**Figure 5.8:** SA-FEC ECC Selection Indoor Link

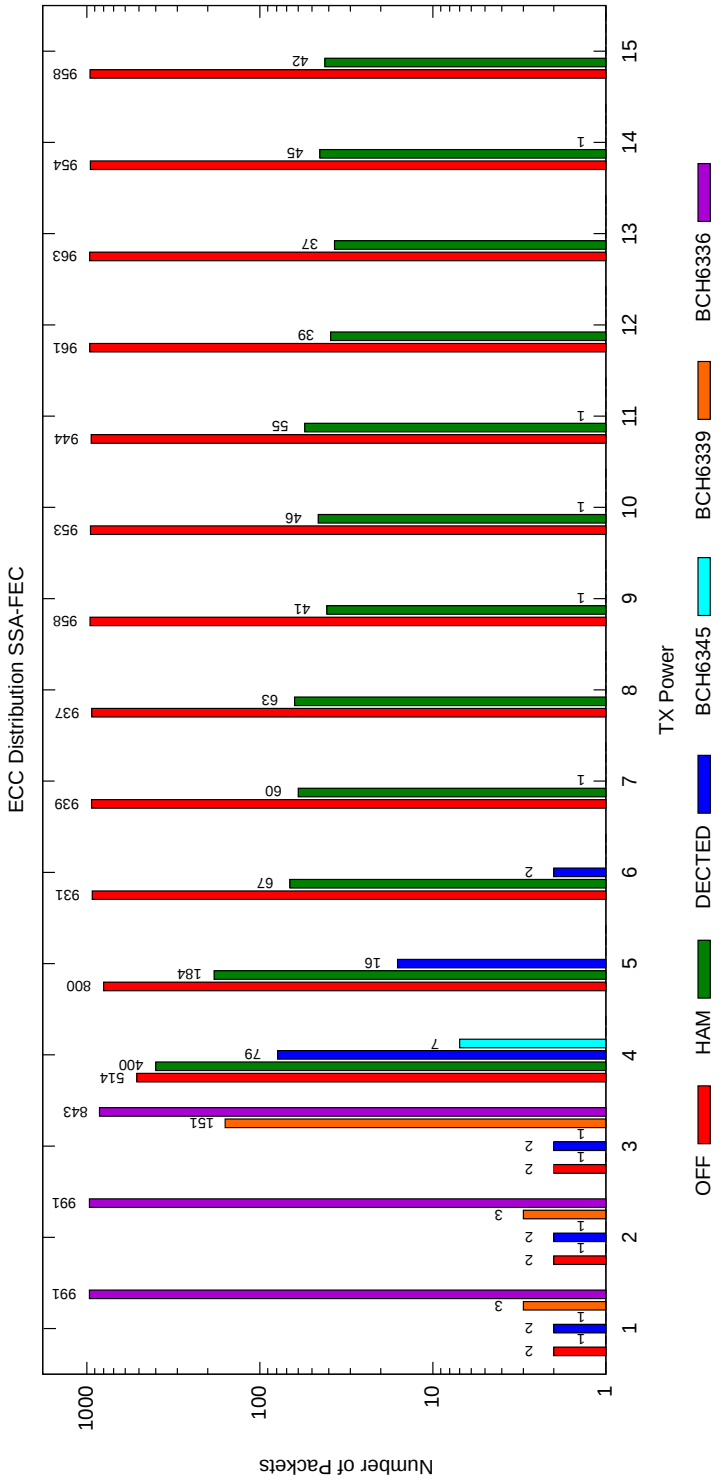



**Figure 5.9:** SSA-FEC ECC Selection Indoor Link

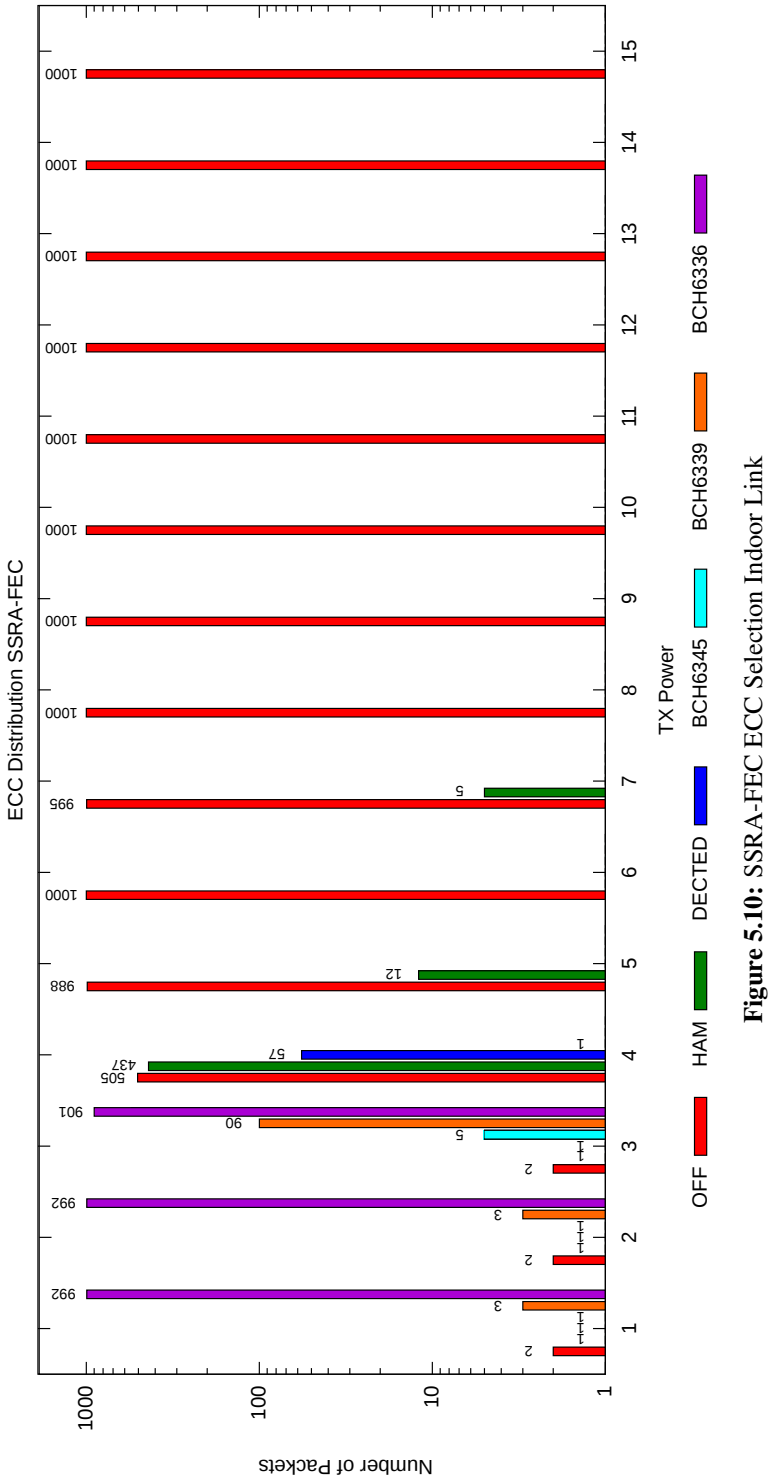



**Figure 5.10:** SSRA-FEC ECC Selection Indoor Link

### Error Occurrence Patterns

An interesting research question we answer in this section is the question what kind of errors actually occur. Therefore, we counted the number of errors per payload of an ECC packet. In this section, we investigated the case where the application of ECC is turned off. The payload consists of a predefined data sequence of 32 bytes that is known to the sender and the receiver. This allows the receiver to compare the received payload with the data, that was sent. Figure 5.11 presents the occurrence of 0 to 10 errors per payload counted over all measured transmission power settings (1 tick (≈ -25 dBm) to 15 ticks (≈ -3.5 dBm)) and applying no ECC (OFF). The data for this evaluation is extracted from the experiment described in Section 5.3.1. Here, only non-encoded ECC packets are considered. To count the occurred errors per ECC packet, the sender transmitted a predefined payload to the receiver, which then compares it bit-wise with the same predefined payload. Note that the y-axis is in a logarithmic scale. Most errors in the payload of the ECC packets were single bit errors. 249 ECC packets resp. 2.04% of the received non-encoded ECC packets contained one bit error. Less frequent were 2 bit errors. 108 resp. 0.88% of the received non-encoded ECC packets contained two bit errors. One can observe the trend that more bit errors within a 32 bytes payload were far less frequent.

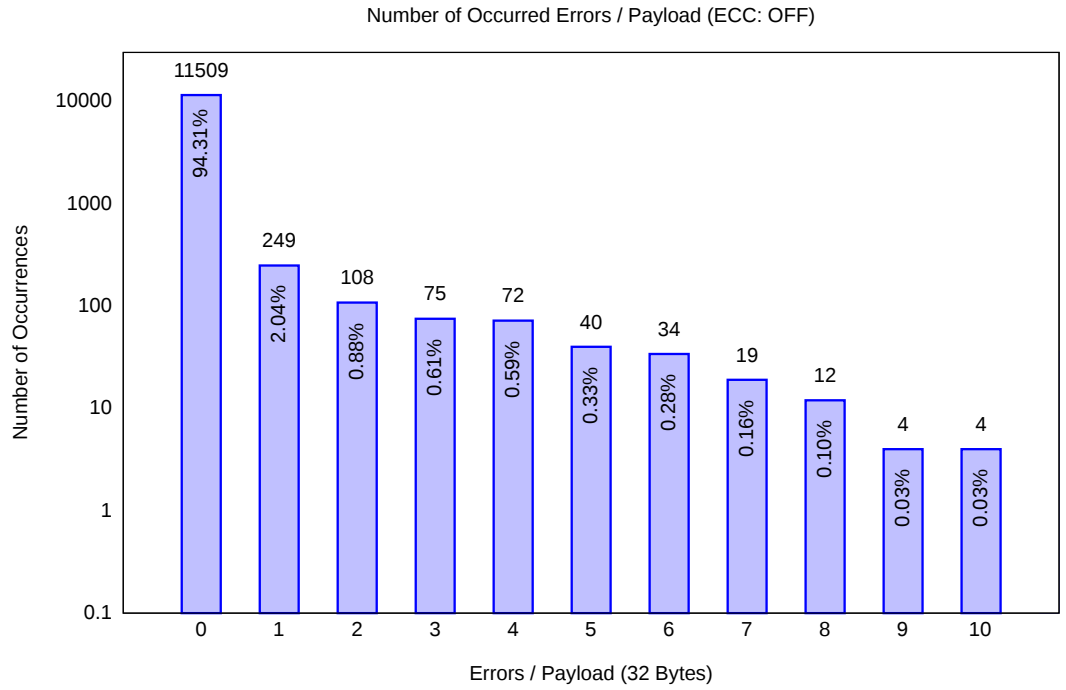



**Figure 5.11:** Occurred Errors

### Corrected Errors

Based on the experimental data of the single-link indoor experiment described in Section 5.3.1, a similar measurement was made for the errors that were corrected across all evaluated ECCs. For this measurement, a histogram of the corrected errors is shown in Figure 5.12. Each column represents the number of ECC packets which contained the number of errors indicated at the x-position of the column. For example, 130 ECC packets contained 10 bit errors that were corrected. In contrast to the scenario above, all implemented static ECCs were considered. The histogram depicts similar results as Figure 5.11. The most frequent corrected payload errors were single bit errors. 2669 resp. 2.49% of the received ECC packets contained a single bit error in the payload. One can observe a similar trend like in Figure 5.11. Again, we cut the error counting after 10 corrected errors per payload.

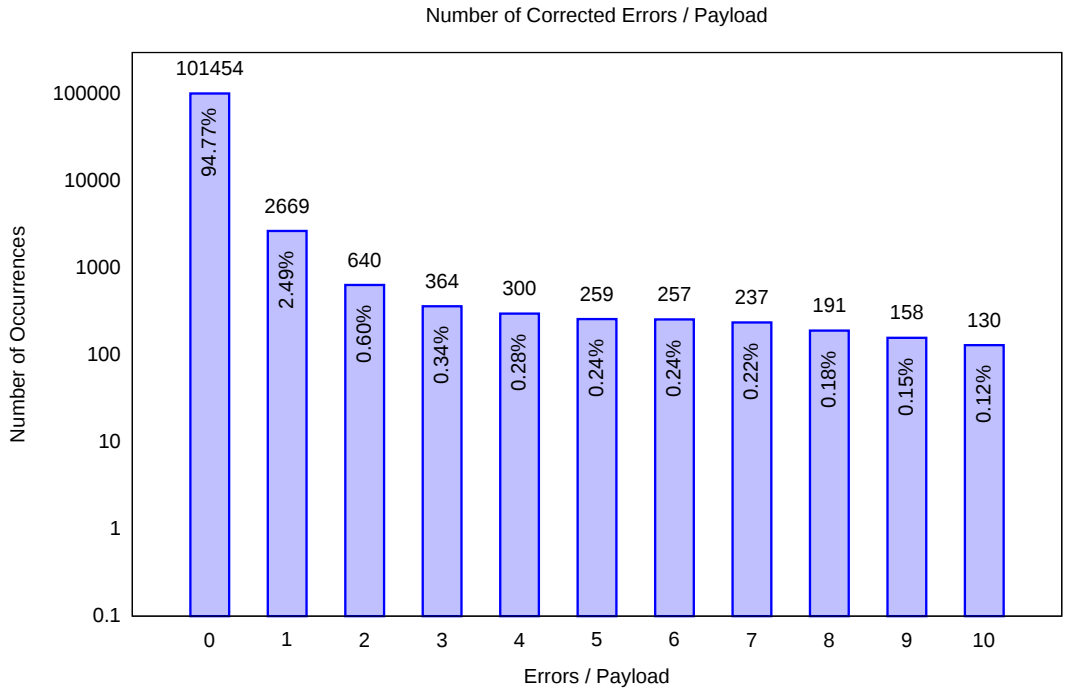



**Figure 5.12:** Corrected Errors

The Figures 5.12 and 5.11 indicate that on the indoor link, the more powerful ECCs that are able to correct more than two bit errors per code word could not use their full correctional potential, because the cases where more than two errors per code word occurred remained very rare ($< 3\%$).

### 5.3.2 Outdoor Single-Link Scenario

The topology for the outdoor link measurements is shown in Figure 5.13. The link consists of a MSB430 sensor node, msb92, placed in room 026 of the Institut für Unternehmensrechnung und Controlling, Engehaldenstrasse 4, 3012 Bern (IUC) on the ground floor. Another MSB430 sensor node identified as msb1 was located in the room 314 on the third floor of the IAM building. The nodes had a line of sight connection through two windows on channel 8. The distance between the two nodes were 48.05 m. The settings for the measurements of this link were the same as in in the indoor single-link scenario. Node msb1 transmitted 1000 ECC packets per transmission power setting and implemented ECC (incl. OFF). For each of the A-FEC mechanisms SA-FEC, SSA-FEC and SSRA-FEC, 1000 packets were transmitted. In every configuration, every two seconds, an ECC packet was sent. We assured that the line-of-sight was not influenced by weather condition changes. Since fog or increased humidity absorb radio signals, the measurements were performed during dry weather conditions.

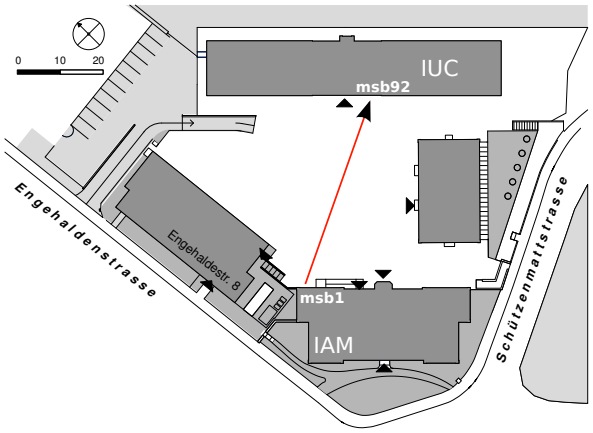


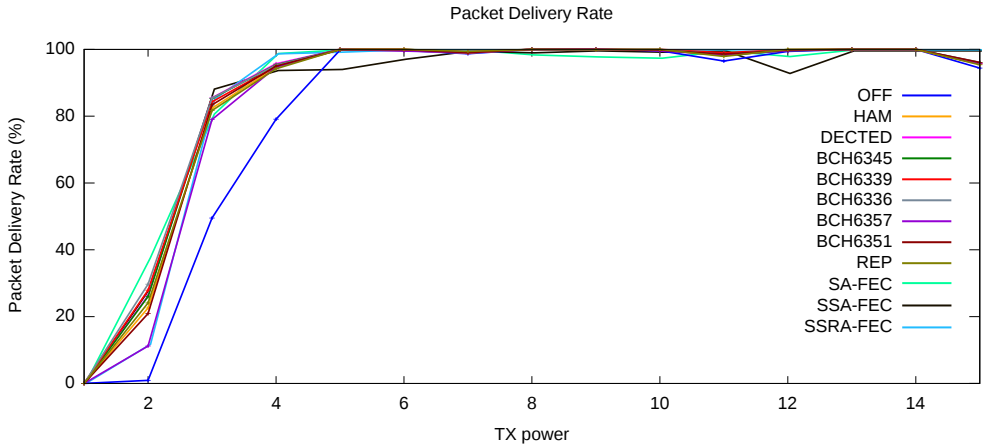

**Figure 5.13:** Outdoor Link

#### Packet Delivery Rate (PDR)

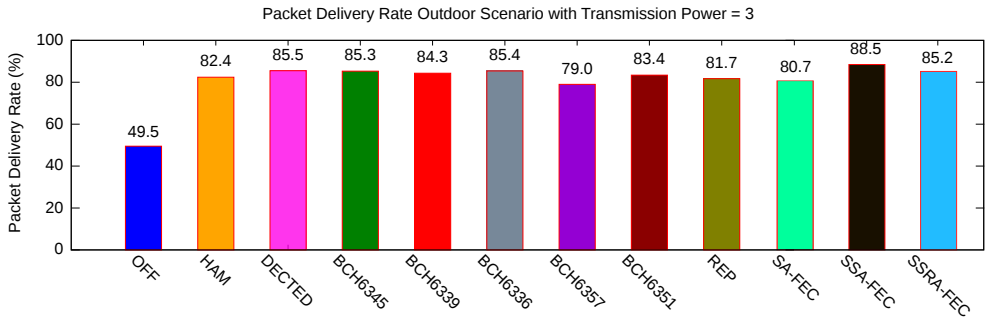
The measurement of the PDR of the outdoor link conveys a similar result as the corresponding measurement of the indoor link (see Figure 5.14(a)). The data for the evaluation resulted from the same experimental setup as described in Section 5.3.2. The application of ECCs incremented the PDR, especially for low transmission power settings as shown in Figure 5.14(b). In general, the PDR values are higher and therefore better than the PDR for the indoor link. The reason for this is most likely the line of sight connection, as opposed to several concrete walls in the indoor link scenario. In contrast to the indoor link, no obstacles were present. Another point to mention is the sharp slope of the graph between the TX power 1 tick ($\approx$ -25 dBm) and 4 ticks ($\approx$ -14.5 dBm). Compared to the slope in the indoor link (see Figure 5.6(a)), the PDR values in the outdoor PDR measurements start to increase dramatically already at a transmission power setting of 1 tick. The reason for this effect is that on the outdoor link, in contrast to the indoor

link, no obstacles such as concrete walls are present. For the outdoor link, the most influence on the PDR comes just from the distance between the nodes, while in case of the indoor link also interferences caused by obstacles influenced the PDR. Besides that, the distance between the nodes of the outdoor links is larger than the distance of the indoor link nodes. Like for the indoor link measurements, the PDR graphs of the ECCs are very close to each other. It seems that the more powerful ECCs did not need to use their full potential, but the increment to the PDR by applying ECCs is significant compared to the case where no ECC was applied.

Besides the static ECCs, the A-FEC mechanisms performed very well. Every A-FEC mechanism achieved comparable PDR values as the static ECCs. Especially, where low transmission powers were used, the A-FEC mechanisms can increase the PDR compared to the case where no ECC is applied. For the case of the transmission power set to 3 ticks, SSA-FEC even achieved the highest PDR.




(a) PDR




(b) PDR with TX Power 3 Ticks ($\approx$ -17 dBm)

**Figure 5.14:** Packet Delivery Rate Outdoor Link

Like in the indoor link scenario, we split up the PDR values of the A-FEC mechanisms for every transmission power setting depending on the applied ECC. One observes that the biggest part of the successfully delivered ECC packets was not encoded at all. Additionally, a similar

pattern for the lower transmission power settings occurred. Using a low transmission power, the A-FEC mechanisms tend to use powerful ECCs. Obviously, the selection of powerful ECCs for example BCH(63,36) in the case of SSA-FEC using TX power set to 2 ticks, was necessary to deliver at least some packets successfully. Otherwise, SA-FEC would have used less powerful ECCs and the part of other ECCs in the column (TX power set to 2 ticks) would have been bigger. The following tables (see Tables 5.4,5.5, and 5.6) present corresponding percentages depicted in the figures above.

| TX Power | OFF | HAM | DECTED | BCH6345 | BCH6339 | BCH6336 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0.2 |
| 2 | 18.9 | 16.1 | 2.1 | 0.4 | 0 | 0 |
| 3 | 79.2 | 1.4 | 0.1 | 0 | 0 | 0 |
| 4 | 98.9 | 0 | 0 | 0 | 0 | 0 |
| 5 | 73.9 | 26.1 | 0 | 0 | 0 | 0 |
| 6 | 97.5 | 2.4 | 0 | 0 | 0 | 0 |
| 7 | 97.4 | 2.6 | 0 | 0 | 0 | 0 |
| 8 | 95.4 | 2.6 | 0.3 | 0.2 | 0 | 0 |
| 9 | 94.7 | 3 | 0.2 | 0 | 0 | 0 |
| 10 | 93.4 | 3.6 | 0.1 | 0.1 | 0.1 | 0.2 |
| 11 | 97.1 | 2.6 | 0 | 0 | 0 | 0 |
| 12 | 94.4 | 3.2 | 0.4 | 0 | 0 | 0 |
| 13 | 97.2 | 2.8 | 0 | 0 | 0 | 0 |
| 14 | 97.2 | 2.8 | 0 | 0 | 0 | 0 |
| 15 | 96.7 | 3.1 | 0.2 | 0 | 0 | 0 |

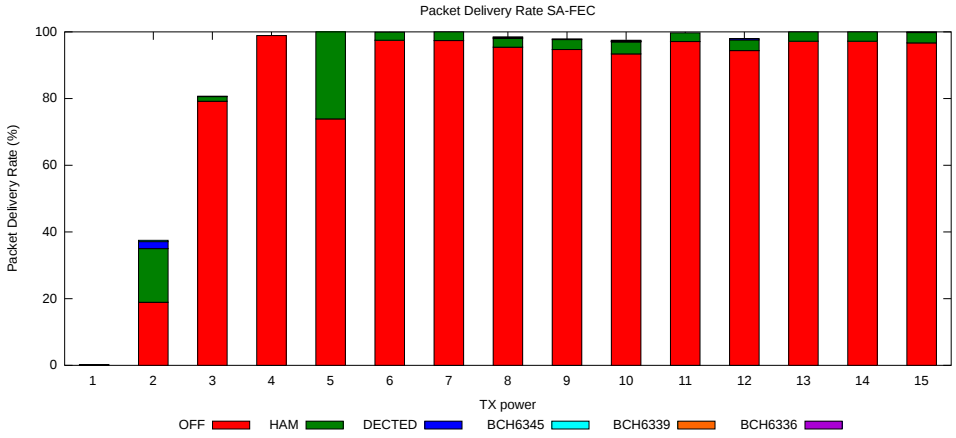
**Table 5.4:** PDR Distribution Outdoor Link SA-FEC

| TX Power | OFF | HAM | DECTED | BCH6345 | BCH6339 | BCH6336 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0.5 |
| 2 | 0 | 0.1 | 0 | 0.1 | 8.2 | 20.4 |
| 3 | 67.7 | 11 | 4 | 2 | 3 | 0.8 |
| 4 | 85.5 | 4.3 | 2.1 | 0.9 | 1.2 | 0.1 |
| 5 | 76.9 | 10.5 | 2.4 | 1.7 | 2.4 | 0.5 |
| 6 | 86 | 8.3 | 2.6 | 0.3 | 0.3 | 0 |
| 7 | 93.7 | 5.4 | 0.8 | 0 | 0 | 0 |
| 8 | 90.5 | 7 | 1.9 | 0 | 0 | 0 |
| 9 | 100 | 0 | 0 | 0 | 0 | 0 |
| 10 | 96 | 3.1 | 0.5 | 0 | 0 | 0 |
| 11 | 100 | 0 | 0 | 0 | 0 | 0 |
| 12 | 73.2 | 6.7 | 3 | 2.8 | 5 | 2.5 |
| 13 | 97.7 | 2 | 0.3 | 0 | 0 | 0 |
| 14 | 97.7 | 2 | 0.3 | 0 | 0 | 0 |
| 15 | 96.3 | 3.2 | 0.5 | 0 | 0 | 0 |

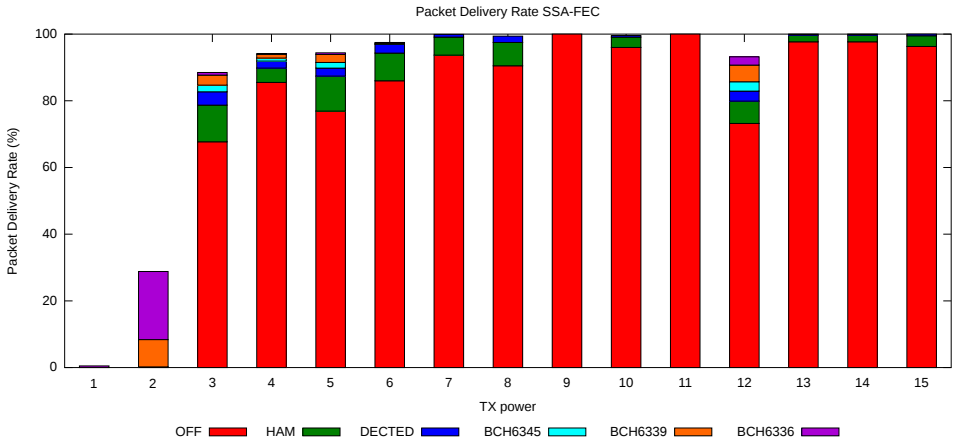
**Table 5.5:** PDR Distribution Outdoor Link SSA-FEC

| TX Power | OFF | HAM | DECTED | BCH6345 | BCH6339 | BCH6336 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0.3 |
| 2 | 0 | 0 | 0 | 0.2 | 1.9 | 9.8 |
| 3 | 76 | 7.8 | 0.9 | 0.4 | 0.1 | 0 |
| 4 | 93.8 | 5 | 0 | 0 | 0 | 0 |
| 5 | 94.8 | 4.5 | 0 | 0 | 0 | 0 |
| 6 | 100 | 0 | 0 | 0 | 0 | 0 |
| 7 | 100 | 0 | 0 | 0 | 0 | 0 |
| 8 | 100 | 0 | 0 | 0 | 0 | 0 |
| 9 | 100 | 0 | 0 | 0 | 0 | 0 |
| 10 | 100 | 0 | 0 | 0 | 0 | 0 |
| 11 | 100 | 0 | 0 | 0 | 0 | 0 |
| 12 | 100 | 0 | 0 | 0 | 0 | 0 |
| 13 | 100 | 0 | 0 | 0 | 0 | 0 |
| 14 | 100 | 0 | 0 | 0 | 0 | 0 |
| 15 | 100 | 0 | 0 | 0 | 0 | 0 |

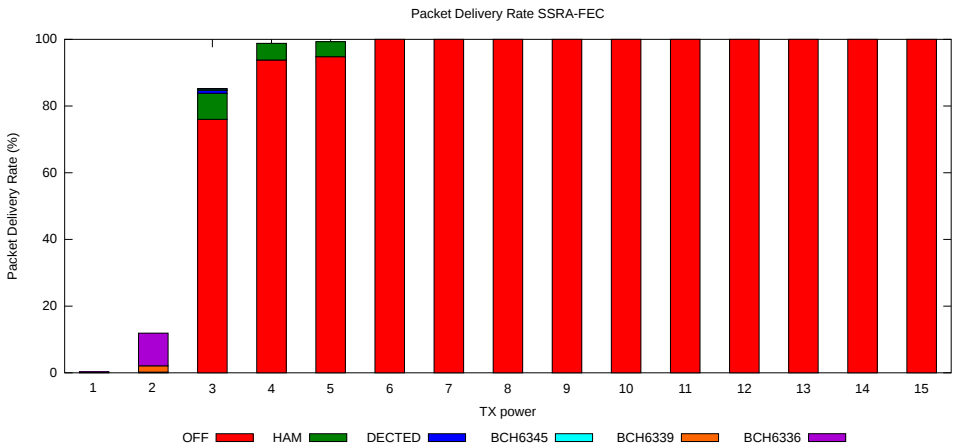
**Table 5.6:** PDR Distribution Outdoor Link SSRA-FEC




(a) PDR SA-FEC




(b) PDR SSA-FEC




(c) PDR SSRA-FEC

**Figure 5.15:** Packet Delivery Rate Distribution Outdoor Link A-FEC

### ECC Selection Behavior

The Figures 5.16, 5.17 and 5.18 illustrate the ECC selection behavior of every A-FEC mechanism used on the outdoor link. The selection pattern is similar to the pattern that resulted from the indoor link measurement. For low transmission powers such as 1 tick and 2 ticks, mostly powerful BCH ECCs were selected. Since the transmission power using these settings was too low to successfully deliver the ECC packets, the A-FEC mechanisms reacted with selecting powerful ECCs. If we compare the selection of the ECCs using the transmission power set to 1 tick with the achieved PDR values for this setting in Figures 5.15(a), 5.15(b) and 5.15(c), one observes that the transmission power was way too low to deliver the packets successfully, even if the most powerful ECC has been applied. This indicates that not the amount of occurred errors was the problem but the simply the low transmission power. It was too low for the packets to reach the receiver.

In general, for most ECC packets no ECC was selected. Obviously, this was the right decision since referring to the PDR values in Figure 5.15, most successfully delivered ECC packets were not encoded.

The run using of SSA-FEC (see Figure 5.17) must have been quite affected by interferences resulting in bit errors since it selected more powerful ECCs compared to the other A-FEC mechanisms for the same transmission power settings.

Comparing the ECC selection behavior on the outdoor link with the ECC selection behavior on the indoor link one notices that on the indoor link, it needed higher transmission power settings (4 ticks) before the A-FEC mechanisms selected for most ECC packets no ECC. In the case of the outdoor link, the A-FEC mechanisms already selected less powerful ECC (such as Hamming(7,4) and DECTED(16,8)) or even no ECC for most ECC packets for lower transmission power values (2 - 3 ticks). The reason for this is that in the indoor link scenario, the radio signal is absorbed by walls and other obstacles, which leads to a weaker radio signal at the receiver and the signal is more vulnerable to distortions. Because of the line of sight between the nodes of the outdoor link, no such obstacles could influence the signal likewise.

Another remark is that on the outdoor link, the variability of the ECC selection is bigger than on the indoor link. Meaning, the ECC selection patterns on the outdoor link are not as consistent as in case of the indoor link. Especially, the selection behavior of SSA-FEC (see Figure 5.17) on the outdoor link stands out compared to SA-FEC and SSRA-FEC. A reason for this behavior could be that for the outdoor link the environmental conditions can change rapidly. Additionally, the variety of signal disturbing factors such as passing electric buses is bigger than for the indoor link, which is completely inside a building that limits the influence of disturbing outdoor factors.

Like in the previous scenario, one can conclude that the application of any A-FEC mechanisms is an advantage. They choose the appropriate ECC to apply for the current link quality and do not use ECCs with high energy costs or complex ECCs wasting their correction potential where they are not necessary.

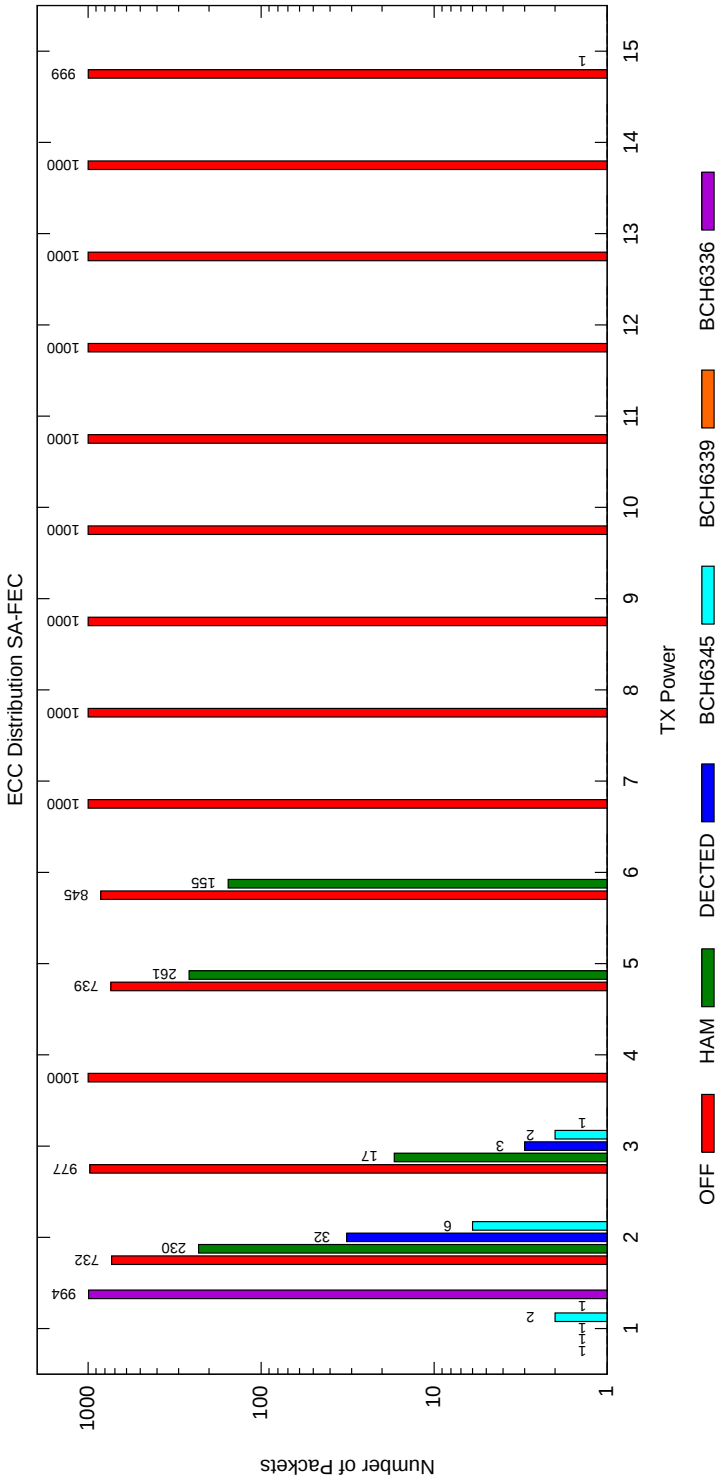



**Figure 5.16:** SA-FEC ECC Selection Outdoor Link

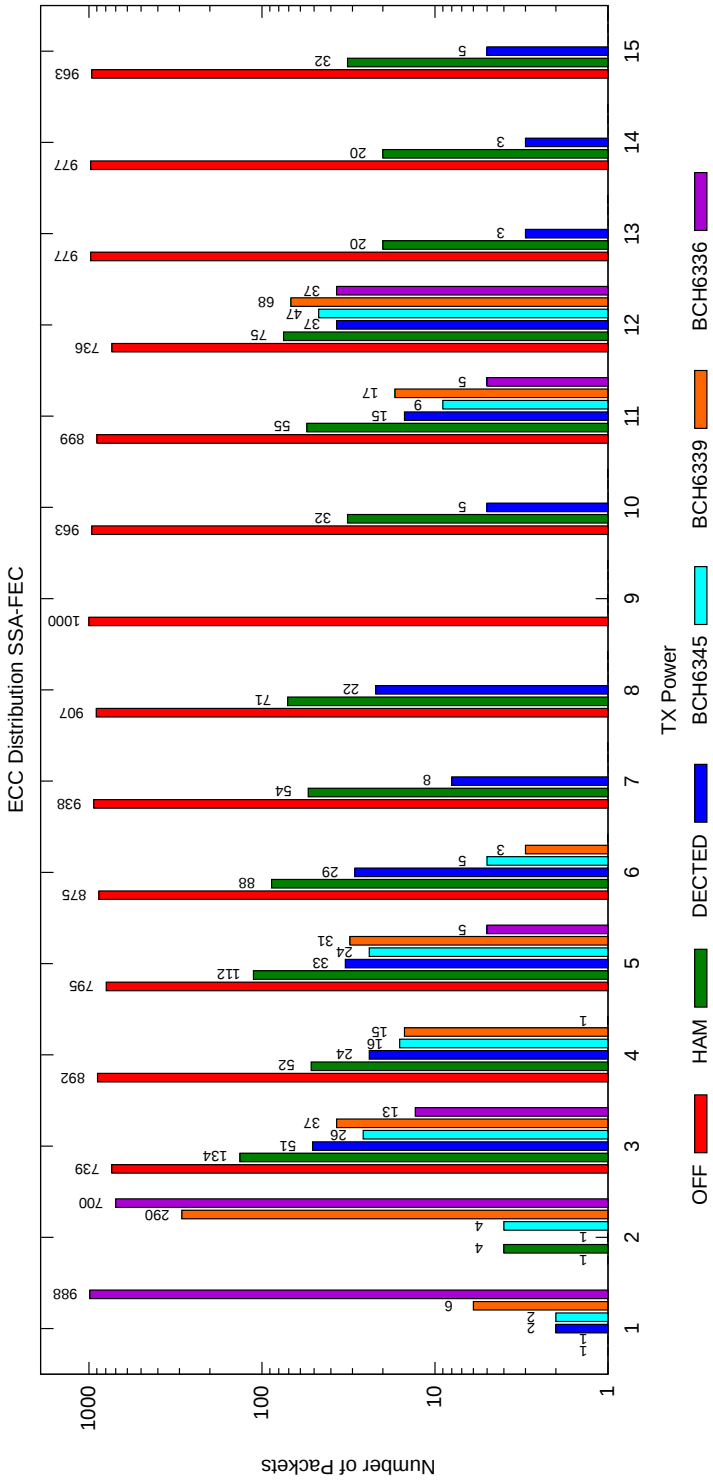



**Figure 5.17:** SSA-FEC ECC Selection Outdoor Link

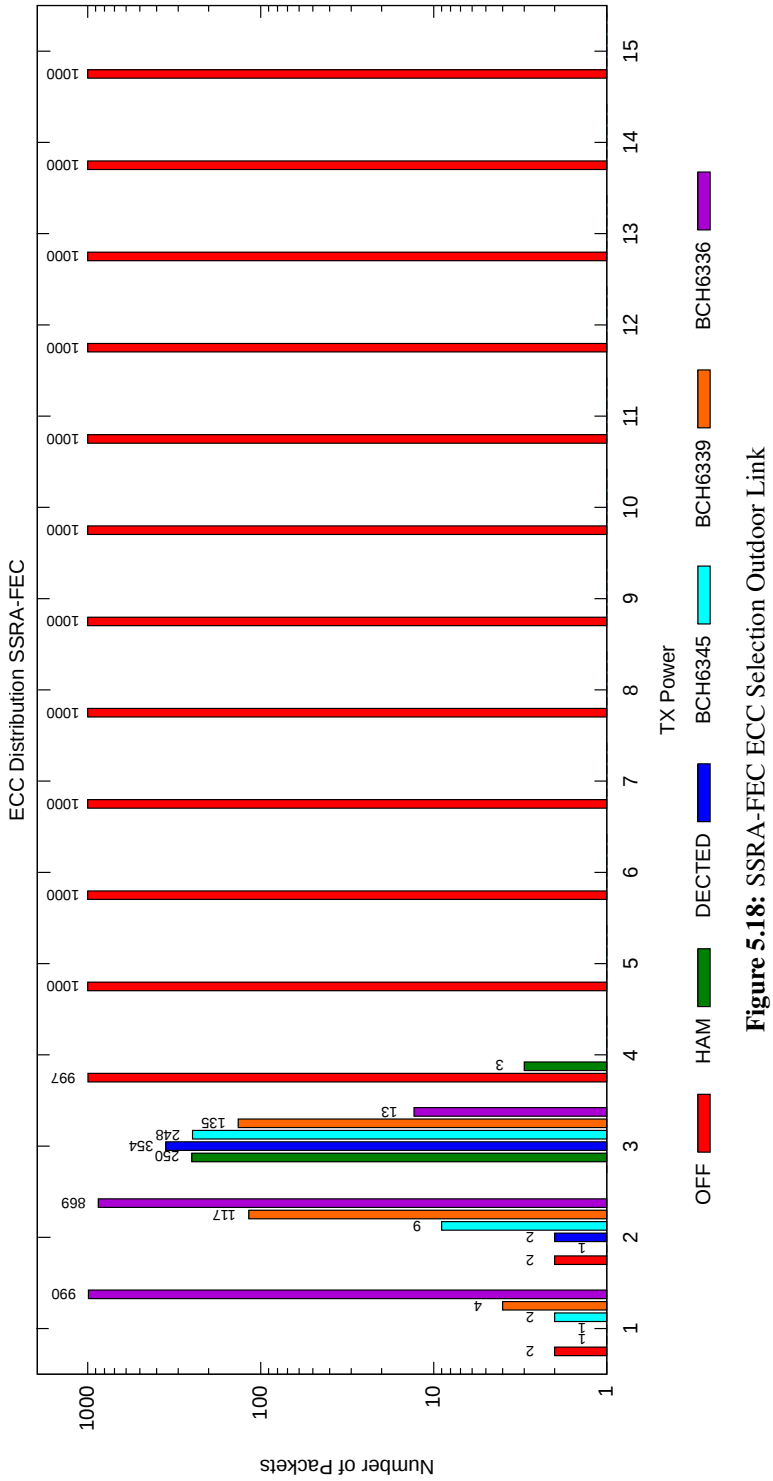



**Figure 5.18:** SSRA-FEC ECC Selection Outdoor Link

### Error Occurrence Patterns

When comparing Figure 5.11 and Figure 5.19, which show the occurred errors in case of applying no ECC, one observes that also in the case of the outdoor link, one and two bit errors were most frequent occurring errors. 407 ECC packets resp. 3.11% of the received ECC packets were disturbed in one bit followed by 141 ECC packets resp. 1.08% of the received ECC packets. As in the indoor measurement, we only consider non-encoded ECC packets. The histogram in Figure 5.19 again only displays up to 10 errors, although very rarely, cases with more than 10 errors in the 32 bytes payload occurred.

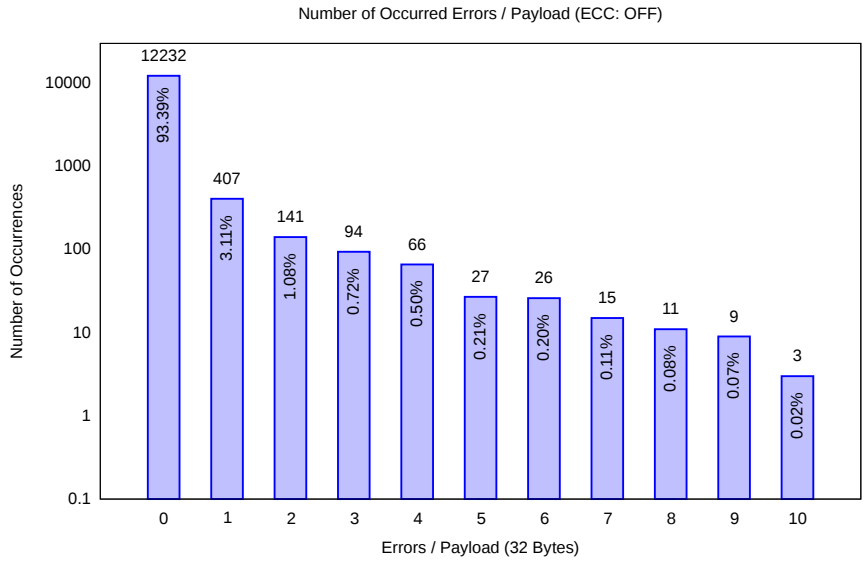



**Figure 5.19:** Occurred Errors

### Corrected Errors

Based on the experimental data of the single-link outdoor experiment described in Section 5.3.2, a similar measurement was made for the errors that were corrected across all evaluated ECCs. For this measurement, a histogram of the corrected errors is shown in Figure 5.20. Each column represents the number of ECC packets which contained the number of corrected errors indicated at the x-position of the column. For example, 0.46% of the received ECC packets contained 4 corrected bit errors in the payload. In opposition to the measurement of the error occurrence pattern described in the previous paragraph, for this evaluation, all implemented ECCs have been considered. In the outdoor scenario, the ECCs were confronted most time with correcting one or two bit errors. This corresponds to the evaluation of the occurred errors. The histogram also looks similar to the one depicting the results of the indoor link (see Figure 5.11) although for the outdoor link, there is a stronger drop of the number of ECC packets which contained more than 3 errors per payload. This indicates that on the outdoor link, the amount of ECC packets corrupted by a small number of errors was higher than on the indoor link.

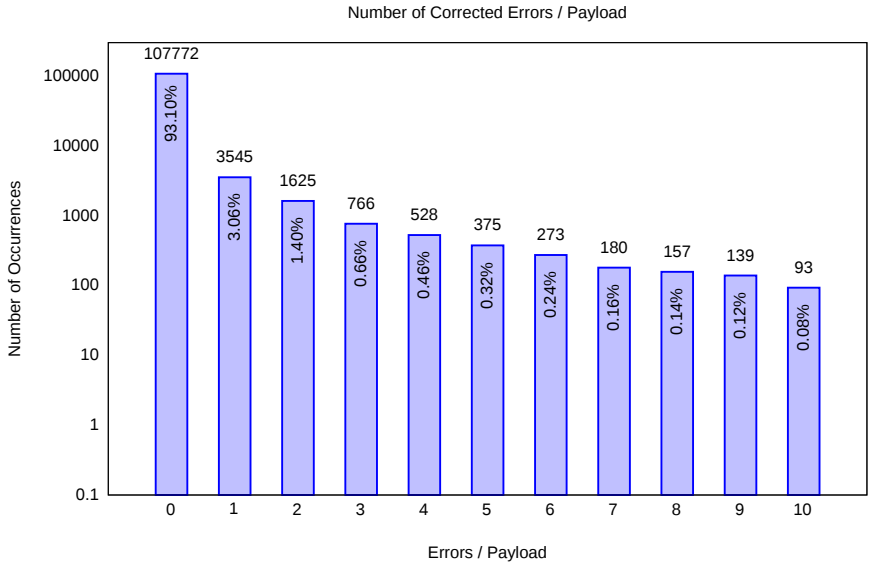



**Figure 5.20:** Corrected Errors

### 5.3.3 Indoor Multi-Link Scenario with Static FEC

To evaluate the behavior of the implemented ECCs over multiple links, the ECCs were tested in the WISEBED testbed provided by the RVS research group using the TARWIS testbed management architecture [41]. The topology is depicted in Figure 5.21. The nodes are distributed over different floors in the IAM building. Each sensor node except the sink node msb3 sends ECC packets. All of the ECC packets are designated for the sink node, located on the top of the topology. Only nodes msb2 and msb6 send directly to the sink node. The remaining nodes need to use at least one additional hop to reach the sink node. The nodes msb2, msb1, msb6, msb7 are called forwarders. The packets of the nodes without direct communication with the sink node are forwarded via these nodes. However, forwarders do not only forward packets, they also send own packets. We call all the nodes (except the sink node) senders. Therefore, forwarders are senders too.

If a forwarder receives an ECC packet, it forwards the packet via the next hop. For every link, the same transmission power setting (5 ticks $\approx$ -12.5 dBm) and the same channels (channel 3) were used. All links have been statically set and are depicted in the topology in Figure 5.21.

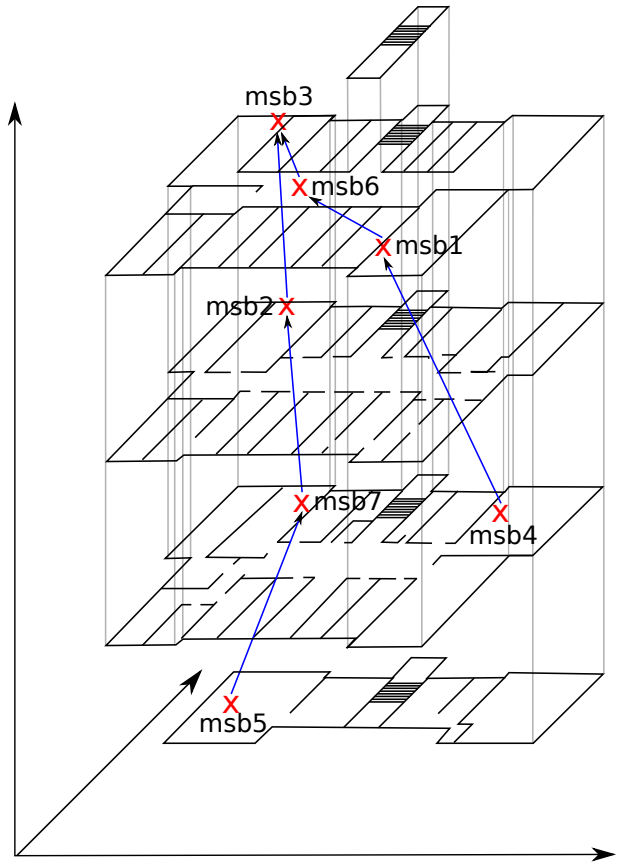



**Figure 5.21:** Multi-Link Topology

For all measurements, each sender generated 1000 ECC packets. The delay between every two ECC packets sent by a single node is chosen according to a uniform random interval of 2 to 4 seconds. On average, a sender hence generates an ECC packet every 3 seconds.

All senders start the sending procedure at the same time. Because of the multi-link topology, every transmission of an ECC packet from a sender triggers subsequent transmissions in the forwarders on the path to the sink node. Hence, the nodes msb4 and msb5 invoke three transmissions in the network for every ECC packet they generate. Therefore, 12 transmissions take place in the entire network every 3 seconds in average, which equals to 4 transmissions per second. With nodes being spatially distributed, not all links however interfere with each other.

The topology naturally leads to an increased usage of the channel near the sink node. The closer the nodes are to the sink node, the more frequent they need to handle packets. In the case of the sink node, msb3, this means that every 3 seconds it needs to handle 6 packets from its neighbors, which equals to 2 packets per second. Since one packet transmission takes roughly 50ms (depending on the codec), we hence obtain a channel utilization of roughly 10% at the sink node.

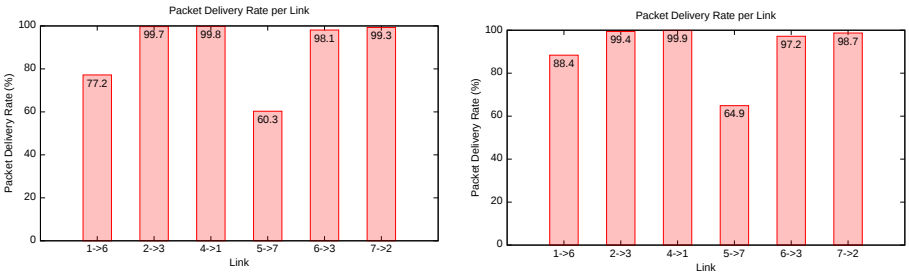
The network implementation of the ScatterWeb OS provides limited packet buffering. This is important considering the application of FEC since the processing time of encoding and decoding of ECC packets in the nodes is significant as described in Section 5.1.1 and 5.1.2. Additionally, before transmitting a packet the MAC layer needs to check if the medium is free. A occupied medium can further delay the emission of a packet by some few tens of milliseconds.

Using these settings and this particular topology, we created an experimental environment, which is close to a real-world application of a WSN. The used parameter values such as the transmission power setting and the ECC packet sending rate were predetermined by empirical measurements in advance in order to obtain network settings with stable but not error-free links and a traffic volume which exhibits a certain collision probability. Collisions can still occur in CSMA-like protocols, since a) random backoff can never prevent collisions but only make them less probable, b) the vulnerable time period when a node switches the transceiver to another state, during which it neither senses nor transmits anything, and c) the hidden node problem which is not addressed at all in the ScatterWeb OS implementation of CSMA.

The following paragraphs evaluate the static FEC experiment. In the static FEC configuration on all links the same ECC was used. Every ECC was tested in a single run, where every sender emitted 1000 ECC packets towards the sink node.

### Packet Delivery Rate (PDR)

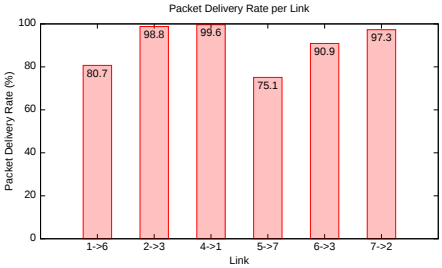
Figure 5.22 depicts the results of the PDR measurement for each ECC and link. All the ECC packets that were sent via any link, including forwarded packets were taken into account. The main focus lies on the links from msb1 to msb6 and msb5 to msb7. Without applying any ECC these two links performed worst (see Figure 5.22(a)). Obviously, these two links suffered more from the impact of signal distortions caused by concrete walls / floors than other links. Consequently, applying FEC on these links should increase the PDR significantly. Almost every ECC increased the PDR on these links compared to apply no ECC. Only on the link between msb1 and msb6, BCH(63,39) performed worse than using no ECC (see Figure 5.22(e)). Similar to this observation is that for some ECCs there is a drop in the PDR for the links that were quite good when no ECC has been used. A reason for the drop could be that using ECC increases the payload size and therefore the entire packet. This can lead to more interferences which corrupt the packets. Especially bit errors in the header of the packet can decrease the PDR, because the header fields are not protected by any ECC. One observes that BCH(63,45) achieved the best PDR results considering all links, although it is not the most powerful ECC of the selection.



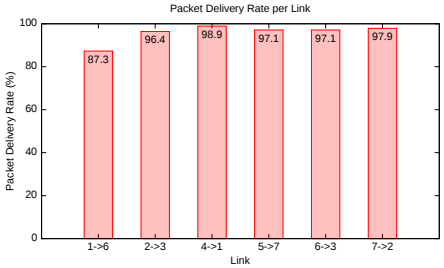

(a) OFF



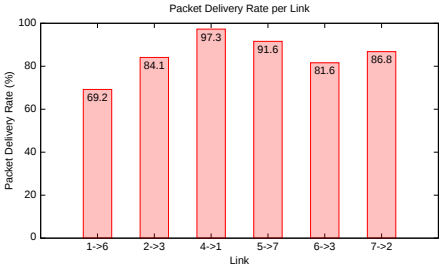

(b) Hamming(7,4)



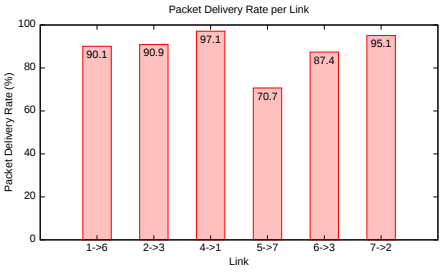

(c) DECTED(16,8)




(d) BCH(63,45)




(e) BCH(63,39)




(f) BCH(63,36)

**Figure 5.22:** Packet Delivery Rate per Link

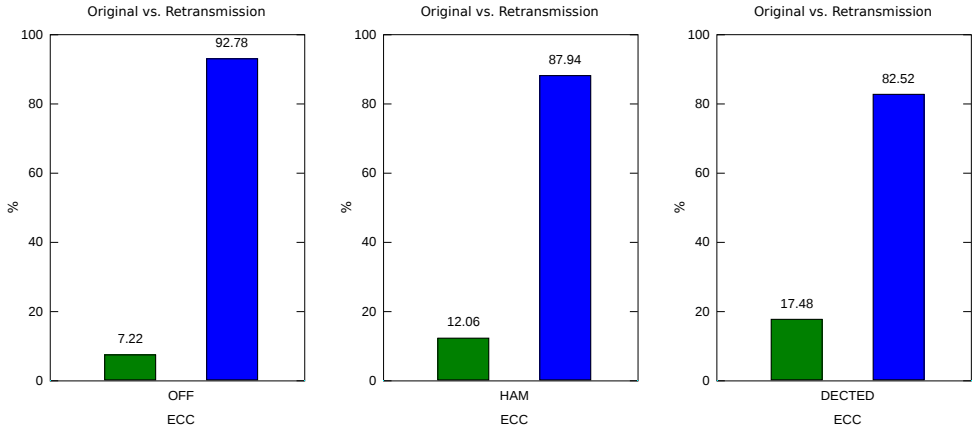
### Retransmissions

We investigated the relation between the amount of retransmitted packets and packets that have been sent originally. Figure 5.23 depicts the relation of original versus retransmitted packets which were received and decoded successfully. The sum of the blue and the green column within a single histogram is equal to 100% and corresponds to all successfully received ECC packets encoded with the corresponding ECC. We define an original packet as a packet that is not a retransmitted packet. A retransmission becomes necessary if an original packet has not been delivered successfully and no ACK has been received. In our implementation after one failed retransmission attempt, the transmission of the packet is aborted and the sender continues with the next packet. In this case the packet could not be delivered to its destination. As a consequence, a receiver either receives an original ECC packet or its retransmission, which is automatically encoded with the next stronger ECC.

One can see in Figure 5.23 that most packets that were received and decoded correctly are original packets. Going deeper into the details, we see that introducing ECC leads on one hand to a higher amount of successfully delivered retransmissions. But on the other hand, the amount of successfully delivered original packets decreases. This could be an indicator for heavier load on the network. Since more additional overhead through the parity information is introduced, the packets are longer and the probability for collisions with other packets increases. Such collisions can either destroy a packet or interfere with it. These interferences may cause bit errors. Some of these might be corrected in an original packet by the ECC. Others are that much corrupted that the sender needs to retransmit the packet. Therefore, the amount of retransmissions increases.

Figure 5.23 may also indicate that the retransmissions are not invoked because of environmental interferences but because of the collisions within the topology caused by the additional network load. This explanation also uncovers the fact, that the usage of a complex ECC is not always appropriate. The additional overhead introduced by the ECCs might also be the reason why the PDR values for BCH(63,36) and BCH(63,39) are not as good as the PDR of BCH(63,45) (see Figures 5.22(f), 5.22(e), 5.22(d)).

Unfortunately, we can not distinguish between the amount of retransmissions caused by collisions and the retransmissions triggered by other reasons, such as interferences with other devices in the ISM band, or simply the absorption of the radio signal by concrete walls and floors. Therefore, it is not possible to explicitly calculate the amount of collisions occurring withing the topology.



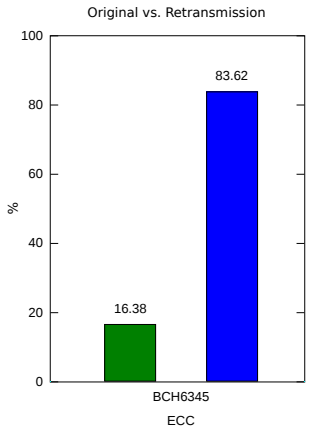

(a) OFF



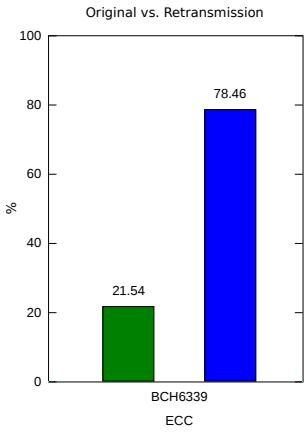

(b) Hamming(7,4)



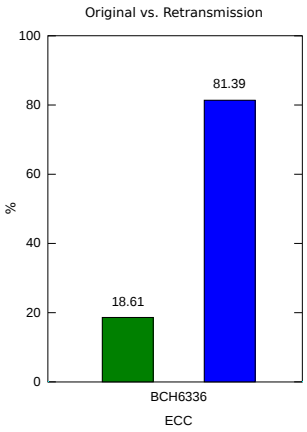

(c) DECTED(16,8)



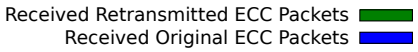

(d) BCH(63,45)




(e) BCH(63,39)




(f) BCH(63,36)




(g) Legend

**Figure 5.23:** Original vs. Retransmission

### Corrected Errors

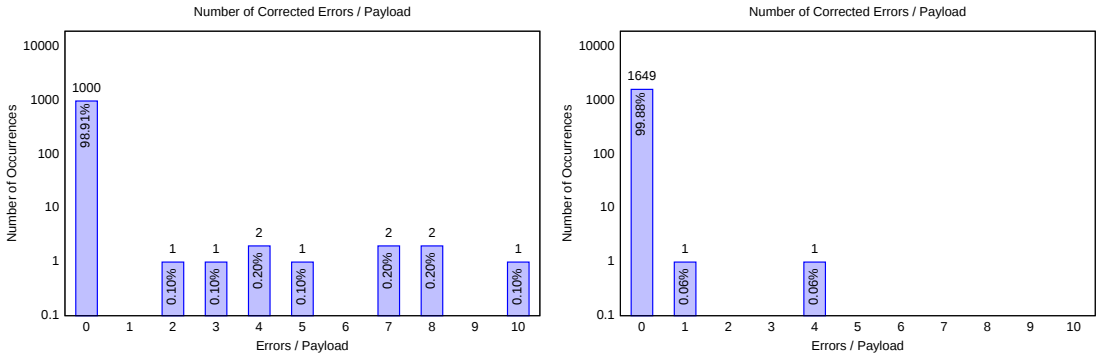
Since in the multi-link scenario there are transmissions that interfere with others, it is interesting to analyze what kind of errors need to be corrected. Therefore, the same evaluation of the corrected errors as in the single-link scenario has been done. The following histograms show the number of corrected errors per payload that the five receiver nodes corrected. More than 10 errors are cut off from the histogram, since the frequency of their occurrence is negligible.

Figure 5.24 depicts the corrected errors of the run using the Hamming(7,4) code. One can see that the errors corrected by nodes msb1, msb2, msb3 (see Figures 5.24(a), 5.24(b), 5.24(c)) are only sparse. Especially msb2 and msb3 did not have to use the potential of the Hamming(7,4) code. Supposedly, these errors were caused by short interferences and therefore, caused only few bit errors that could be corrected with Hamming(7,4). For the nodes msb6 and msb7 (see Figures 5.24(d), 5.24(e)), the trend that more errors per payload occur less than single bit errors is well visible.

The results of the error measurement for the DECTED(16,8) code are illustrated in Figure 5.25. The results resemble the Hamming(7,4) code results. The nodes msb2 and msb3 corrected almost no errors. This indicates that the links to these nodes were quite good. Referring to the PDR values of the links which have msb2 or msb3 as receivers, this assumption can be confirmed. This also holds for the link from msb4 to msb1.

Like in the single-link error measurements, single bit and double bit errors were the most occurring errors. This observation is independent from the topology of the sensor network and the used ECC. If one compares the corrected error statistics of all the multi-link experiments, it can be concluded that the number of corrected errors is a per-link issue and that, as a conclusion, ECCs should be applied and tailored on a per-link basis. For example, node msb6 and msb7 corrected more errors than the other nodes independently from the ECC that they used (see Figures 5.24(d), 5.25(d), 5.26(d), 5.27(d), 5.28(d)). Considering the low PDR values of the links where msb6 and msb7 represent the endpoints (see Figure 5.22), it can be stated that the links from msb1 to msb6 and from msb5 to msb7 were quite affected by bit errors. The reason for this probably lies in the topology. It might be that the many walls that these links need to cross, absorb more of the signal than the walls / floors which intercept on the other links. It is also possible that there were more interferences with other radio waves on the link to msb6 and msb7. Therefore, it is most likely a topological effect. Consequently, msb6 and msb7 corrected more errors than the other nodes.

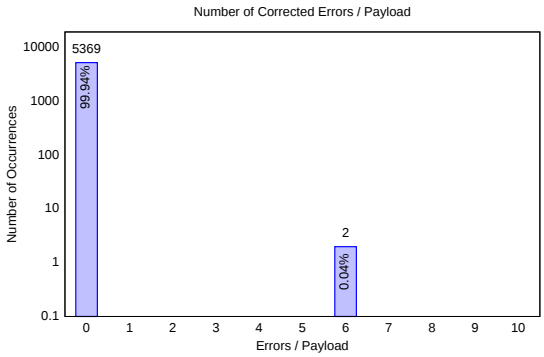
One additionally observes that there are some isolated errors showing up, for example in Figure 5.24(c) or 5.26(a). Since we perform our experiments in a real-world environment, such error correction patterns are highly possible through short-lived interferences. Note that we use a logarithmic scale on the y-axis. This makes the isolated columns appear more important than they are. In reality, the few outliers in Figures 5.24(a) 5.24(b) 5.24(c) where few errors occurred were statistically much less significant than in the case of msb6 and msb7 in Figures 5.24(d) 5.24(e). Moreover, for the evaluation we directly used the raw data from the experiments. The data was not processed before doing the evaluation. In general, we have to mention that results from such a real-world experiment are very difficult to reproduce, because it is almost impossible to create the same environmental conditions again [35]. There are environmental influences which can not be controlled such as the behavior people within the building.



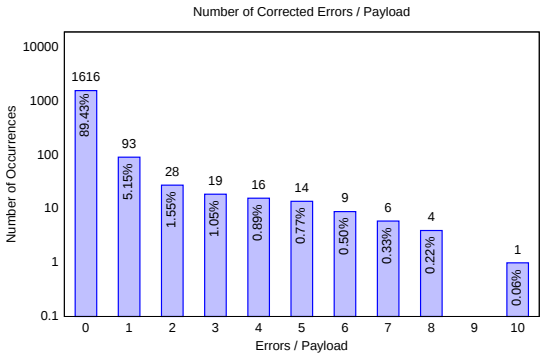

(a) msb1



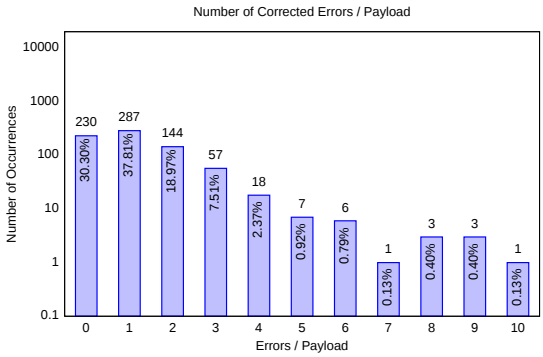

(b) msb2




(c) msb3

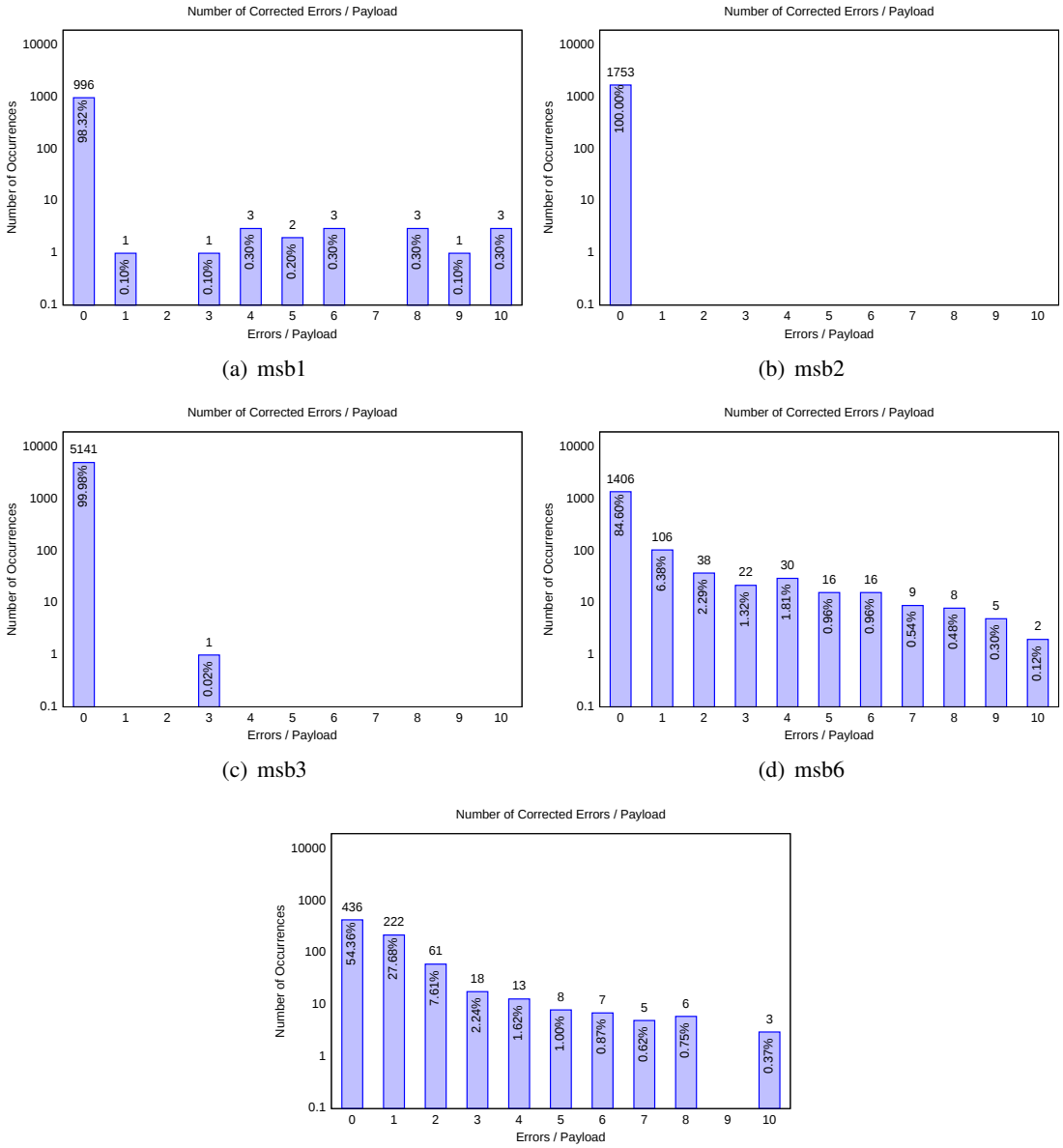



(d) msb6




(e) msb7

**Figure 5.24:** Hamming(7,4): Corrected Errors




(a) msb1

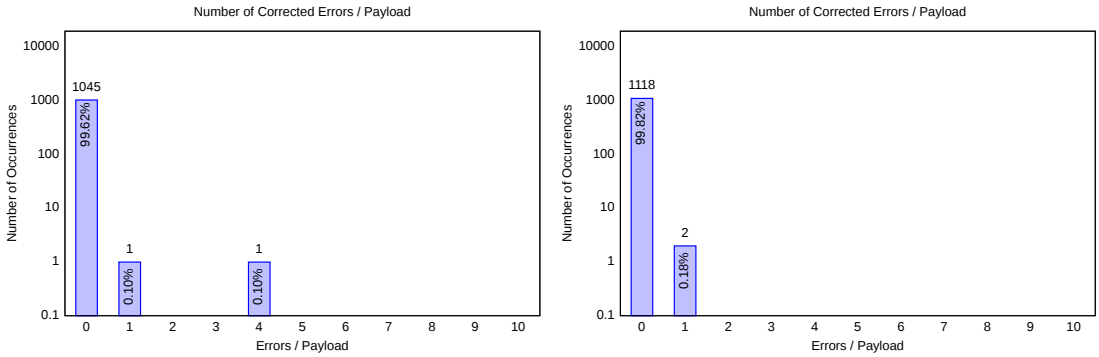
(b) msb2

(c) msb3

(d) msb6

(e) msb7

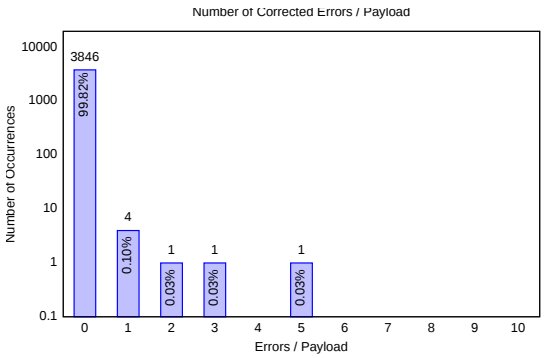
**Figure 5.25:** DECTED(16,8): Corrected Errors



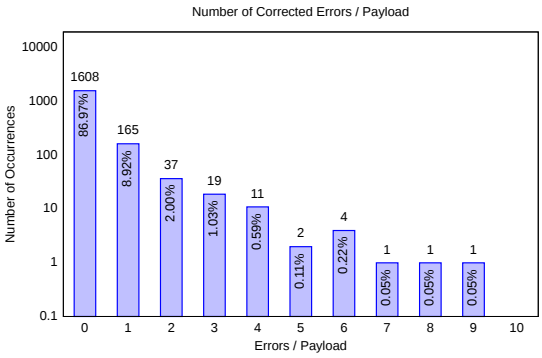

(a) msb1



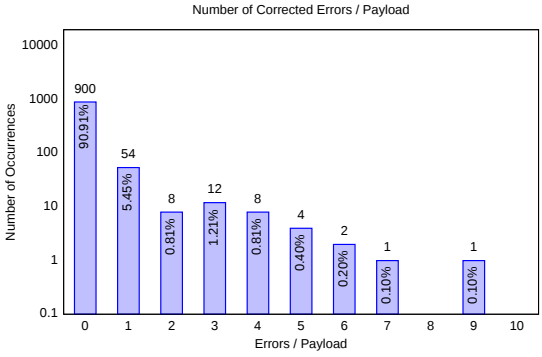

(b) msb2




(c) msb3

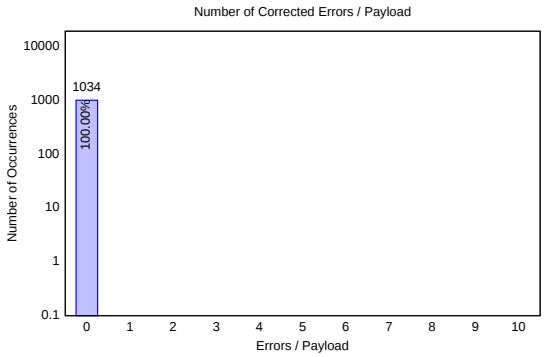



(d) msb6

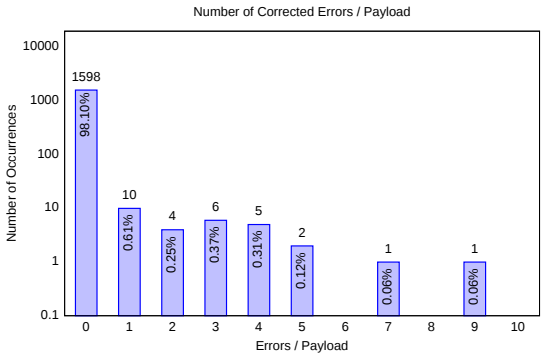



(e) msb7

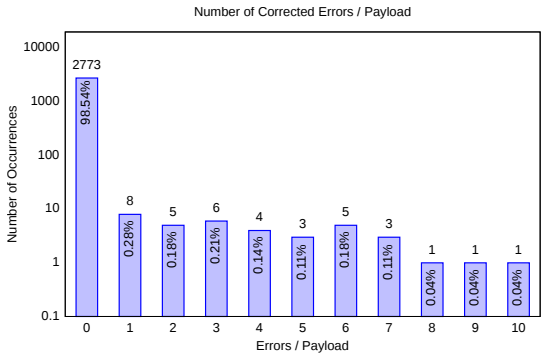
**Figure 5.26:** BCH(63,45): Corrected Errors



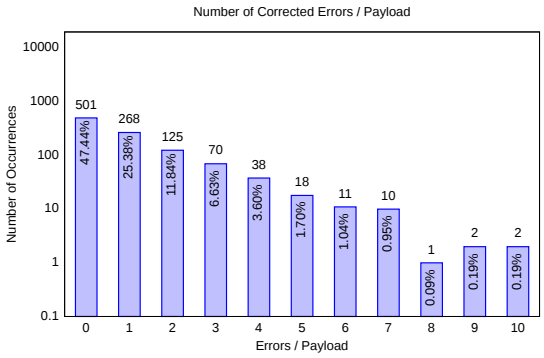

(a) msb1



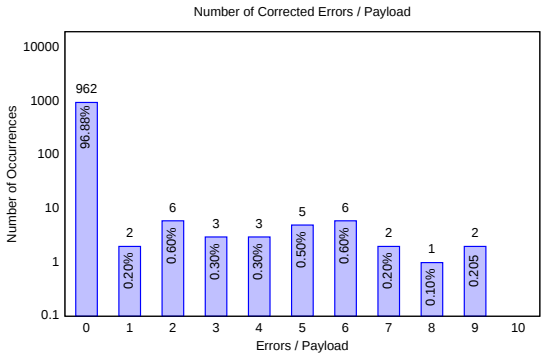

(b) msb2




(c) msb3

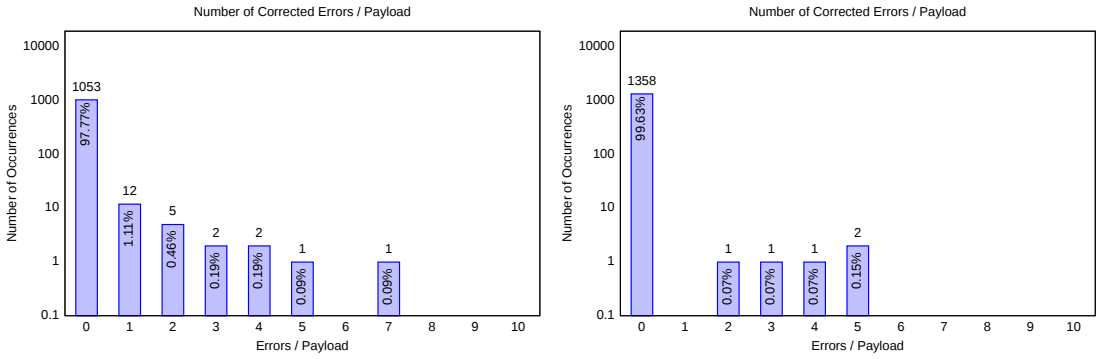



(d) msb6




(e) msb7

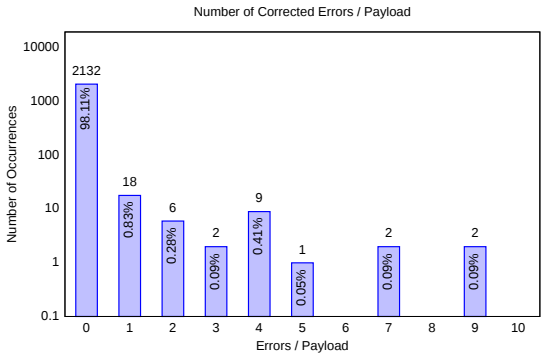
**Figure 5.27:** BCH(63,39): Corrected Errors



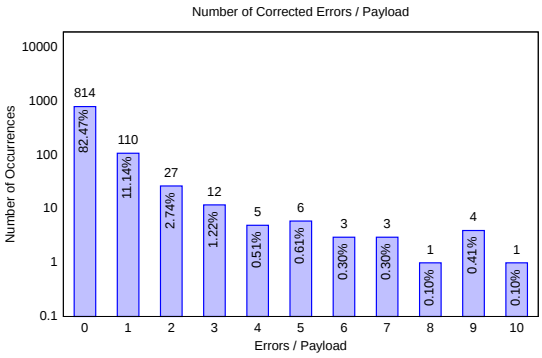

(a) msb1



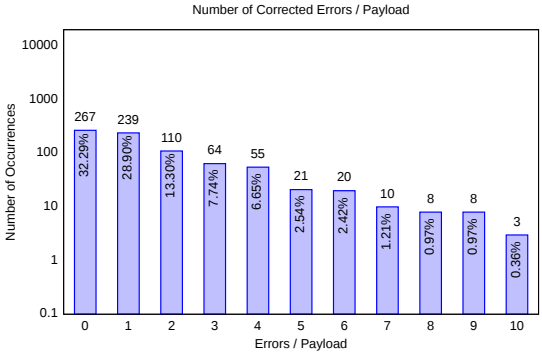

(b) msb2




(c) msb3




(d) msb6




(e) msb7

**Figure 5.28:** BCH(63,36): Corrected Errors

### 5.3.4 Indoor Multi-Link Scenario with Adaptive FEC

We evaluated the A-FEC mechanisms in the same multi-link topology as used for the static ECCs. The experiment setup was the same as described in Section 5.3.3. The only difference is that the nodes are configured to use one of the three different A-FEC mechanisms, instead of the fixed ECCs. Every sender initially started to send ECC packets using no ECC. During the experiment runtime, the senders of every link independently began to switch between the ECCs depending on the link quality. For every A-FEC mechanism, each sender emitted 1000 ECC packets with one packet every 3 seconds in average. The evaluated adaptive approaches are the Stateless Adaptive FEC (SA-FEC), the Stateful Sender Adaptive FEC (SSA-FEC) and the Stateful Sender Receiver Adaptive FEC (SSRA-FEC) mechanism. SA-FEC only takes the last ECC into account to select the ECC for the next transmission. SSA-FEC is one of the two history-based approaches. It considers a history of the last five ECCs. SSRA-FEC represents the extension of SSA-FEC. The SSRA-FEC additionally considers a history of the maximum corrected errors per block of the last five transmissions. The history-based approaches use averaging to select the next ECC. A detailed description of the evaluated A-FEC approaches is presented in Section 4.2.

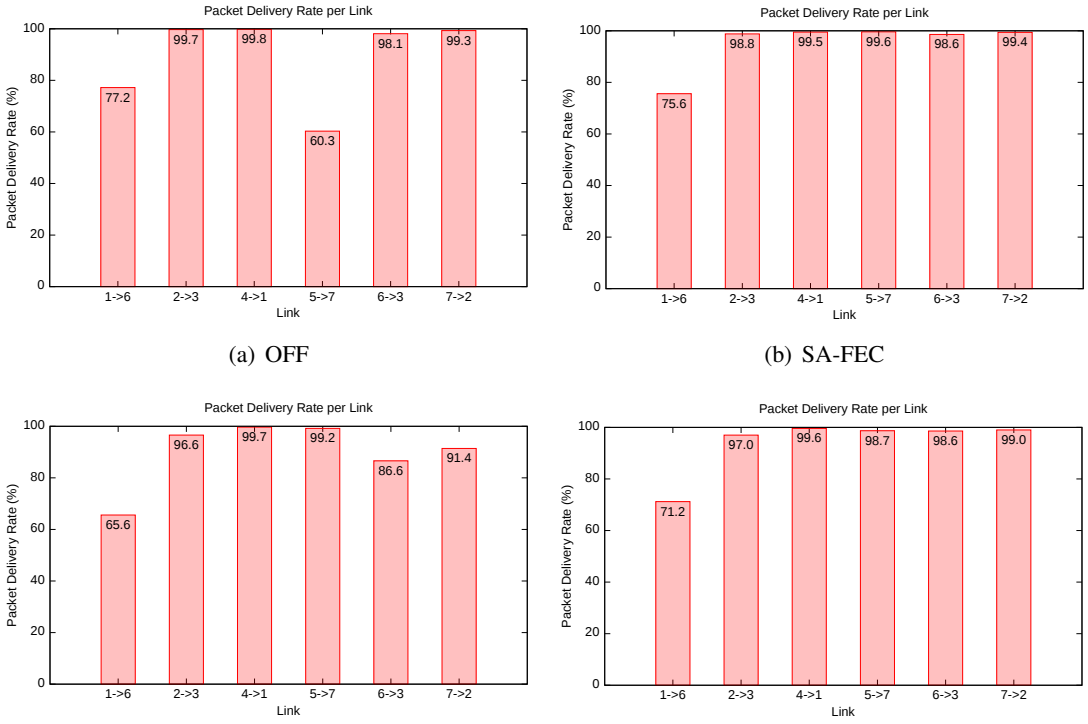
#### Packet Delivery Rate (PDR)

We expected that the PDR of the adaptive FEC approaches would achieve the best results, because adaptive FEC schemes adapt themselves to the environmental circumstances, for example to link quality changes which might be happening quite quickly. The evaluation shows that the adaptive FECs achieve good results (see Figure 5.29). In general, the links were more reliable using adaptive FEC than using no ECC. This is a sign for the adaptation mechanisms to work quite well. Especially, the link from node msb5 to msb7 benefited from the adaptive FEC. The PDR on this link was even higher than using any static ECC. On the other hand, the link from msb1 to msb6, could not benefit from any A-FEC mechanism considering PDR, although the results are comparable with the PDR values using no ECC. In general, the A-FEC mechanisms performed similar to the BCH(63,45) static ECC, which performs best compared to all static ECCs. To achieve a similar performance as BCH(63,45), one should choose an A-FEC mechanism. The idea behind this preference is that A-FEC mechanisms probably consume less energy than complex and powerful static ECCs such as the BCH(63,45), BCH(63,39), BCH(63,36). As seen in Section 5.2, the longer a encoding and decoding process takes, the more energy is consumed. Complex and powerful ECCs need more time to encode and decode the payload than simpler ECCs and are therefore more expensive in terms of energy consumption. Therefore, the usage of A-FEC mechanisms that determine the cheapest ECC for the current link quality and still achieve comparable PDR results as complex static ECCs is a benefit.

The A-FEC mechanisms SA-FEC and SSRA-FEC performed on the same level. Possible reasons why the performances of SA-FEC and SSRA-FEC are on the same level could be on one hand that short-lived interferences which corrupted the payload of the ECC packets may be compensated by retransmissions, which are protected through a more powerful ECC than the original ECC packet. On the other hand, it is possible that most packets were not harmed. However, there is a increase of the PDR in SA-FEC on link from msb1 to msb6 of 4.4%. Moreover,

most PDR values for the other links are higher in SA-FEC than in SSRA-FEC. From comparing all three approaches together, we conclude that the history-based A-FEC approaches can not point out any advantage of the long-term knowledge stored in the histories. Therefore, one should choose the most flexible SA-FEC to increase the PDR.

If we compare the A-FEC mechanisms with the case where no ECC is applied to the ECC packets, one notices that especially the increment in the PDR for the link msb5 to msb7 is significant. Obviously, on this link errors occurred, which the A-FEC mechanisms could correct. Additionally, for links which reach a high PDR without FEC, any A-FEC mechanism represents an achievement considering an increment of PDR.




(a) OFF

(b) SA-FEC

(c) SSA-FEC

(d) SSRA-FEC

**Figure 5.29:** Packet Delivery Rate per Link

### ECC Selection Behavior

The evaluation showed in Figure 5.30 is based on the same raw data from the experiment used to evaluate the PDR and the number of corrected errors. It illustrates the behavior of the A-FEC mechanisms in the entire network. It quantifies the selection of the ECCs by the different A-FEC mechanisms. These results are collected over every link, hence they represent the entire networks' end-to-end ECC distribution. The green column represents the percentage of successfully delivered ECC packet retransmissions compared to the sum of all successfully received ECC packets. It is computed as follows:

$$Retrans(x) = \frac{rcv\_Retrans(x)}{rcv\_Retrans + rcv\_Original}$$

Where $rcv\_Retrans(x)$ is equal to the successfully received retransmissions encoded with ECC $x$. The number of all successfully received original ECC packets is represented by $rcv\_Original$. The variable $rcv\_Retrans$ is equal to the number of all successfully received retransmitted ECC packets. The sum of $rcv\_Retrans$ and $rcv\_Original$ is equal to all successfully received ECC packets in the entire network independently from the applied ECC. The orange column represents the percentage of the successfully delivered original ECC packets compared to the sum of all successfully received ECC packets. It is calculated as follows:

$$Original(x) = \frac{rcv\_Original(x)}{rcv\_Retrans + rcv\_Original}$$

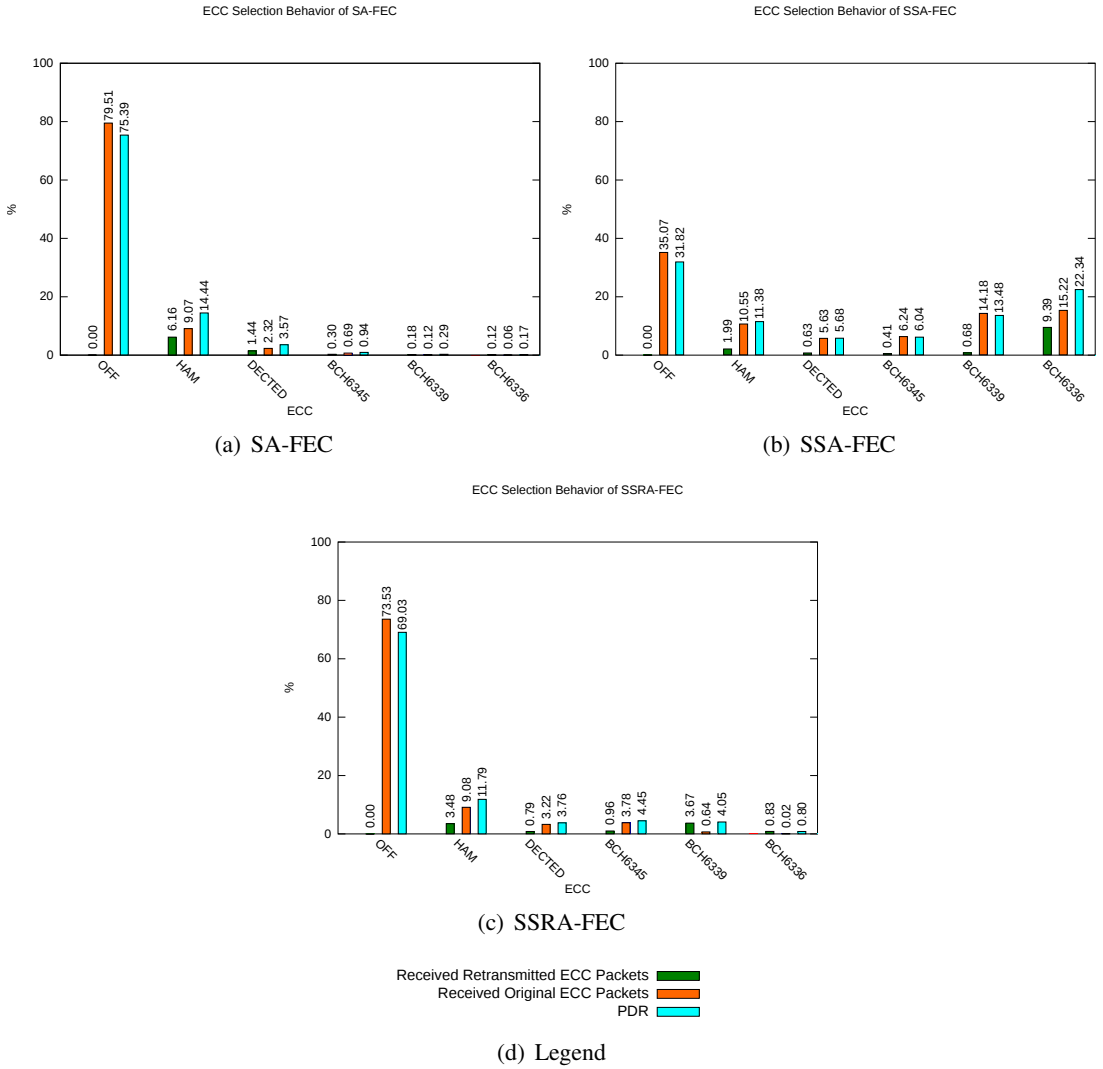
Where $rcv\_Original(x)$ is equal to the successfully received original ECC packets encoded with ECC $x$. The sum of the percentage of every orange and green column per histogram is equal to 100% of all the successfully received ECC packets. The most right column colored in turquoise indicates, which portion of PDR was achieved by which ECC. It is computed as the ratio of successfully received ECC packets per ECC divided by the sum of all sent packets.

$$PDR(x) = \frac{rcv\_Original(x) + rcv\_Retrans(x)}{rcv\_Retrans + rcv\_Original}$$

A general observation is that in most cases, no ECC was selected – especially for the SA-FEC and the SSRA-FEC scheme. This indicates that the topology in general did not suffer from severe interferences which corrupted packets. Except for the SSA-FEC, most ECC packets were not encoded or only with low correction power ECCs such as Hamming(7,4) or DECTED(16,8). This is in agreement with the general observation that the most corrected errors were single or double bit errors. In the case of SSA-FEC, the distribution of the selected ECCs tends more to more powerful ECCs. This phenomena is discussed later in this section, where the ECC selection per link is presented.

One might notice that the percentage of retransmissions using no ECC is zero for all the adaptive approaches. The explanation is simple: As described in Section 4.2, the adaptive FEC mechanisms always use the next more powerful ECC for the retransmission of a packet. Therefore, to every retransmitted ECC packet an ECC is applied that at least corrects one error per code word.

An indicator for the amount of required retransmissions gives the percentage of the received retransmitted ECC packets. The more accurate the adaptation to the link quality is, the less retransmissions because of errors are necessary. Therefore, we can quantify the accuracy of the adaptive FEC mechanisms by counting how many of the received packets are retransmissions. The summation of the percentages of the received retransmitted ECC packets shows that the SA-FEC uses least retransmissions (8.2%). The SSA-FEC and the SSRA-FEC needed more retransmissions: 13.1% and 9.63%, respectively. Therefore, we conclude that the SA-FEC is most accurate in selecting the ECCs. Moreover, since the SA-FEC is not based on averaging to determine the upcoming ECC to apply, it is faster in adaption to link quality changes than the other A-FEC approaches.




(a) SA-FEC

(b) SSA-FEC

(c) SSRA-FEC

(d) Legend

**Figure 5.30:** A-FEC ECC Selection Behavior

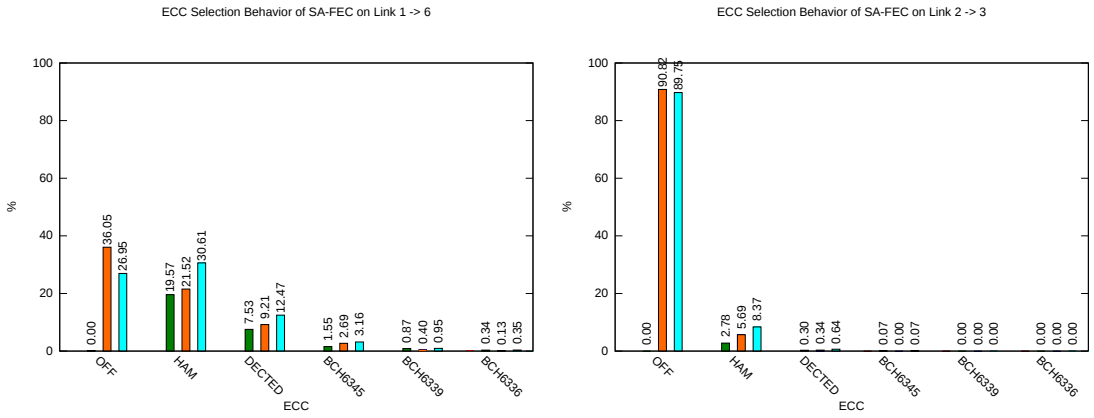
In this paragraph, we go deeper and take a look at the adaptive FEC mechanisms from a link-based point of view. Figure 5.31 shows the distribution of the ECCs for every link using SA-FEC. As one can see, the most selected ECCs are the Hamming(7,4) and DECTED(16,8) code. Most packets were successfully delivered without applying of any ECC. As derived from Figure 5.29(b), the link from msb1 to msb6 is of significantly lower quality that the rest of the links. Therefore, a significant number of packets were encoded using Hamming(7,4) and DECTED(16,8). It is a logical consequence that on this link, more powerful ECC are selected and more retransmissions are triggered. For this link, the Hamming(7,4) ECC achieved the best results.

In Figure 5.32, the results for SSA-FEC is depicted. The situation here is quite different than in the case of SA-FEC. For the links from msb1 to msb6 and from msb2 to msb3 as well as for the link from msb6 to msb3 (see Figures 5.32(a), 5.32(b), and 5.32(e)), SSA-FEC tends to use strong ECCs such as the BCH ECCs. On the other links, the simple ECCs are selected. It is quite interesting that on the link from msb1 to msb6 a lot of retransmissions were needed, although powerful ECCs were selected. This indicates that even when using BCH(63,39) the codec was still too weak to correct all the errors, which means that msb1 sent the ECC packets encoded with BCH(63,36). The fact that most successfully delivered packets to msb6 were retransmitted ECC packets encoded with BCH(63,36), is a consequence of this. Another reason for the increased amount of retransmission could also be that a lot of ACKs were lost. The loss of ACKs trigger a retransmission although the packet was delivered successfully. The loss of an ACK message can happen for two reasons: First, as described in Section 4.2, ACK are not protected against bit errors and second, the ScatterWeb OS does not perform a carrier sensing before sending an ACK. It can happen that if the decoding takes long as in the case of BCH(63,36), another node starts to send since the medium is free, although the receiver has not yet sent the ACK. If the decoding process is successfully finished and other transmissions are taking place, the probability of collisions increases, since the ACKs are sent without a previous carrier sensing. As already described in Section 5.3.3, the amount and the density of transmissions increases towards the sink node. Therefore, the probability of collisions increases on the mentioned links, which again causes more interferences. In contrast, the most successfully delivered packets on the link from msb6 to msb3 were not retransmissions. Therefore, it can be said that for this link not the loss of ACKs caused SSA-FEC to select BCH(63,36), but the errors caused by interferences.

In Figure 5.33, the results for the adaptive FEC mechanism SSRA-FEC are shown. For all links except that from msb1 to msb6, most packets were successfully delivered without using any ECC. Moreover, almost no retransmissions were necessary, because the first transmission attempts were successful. Like in the other measurements, the measurements of the link from msb1 to msb6 looks different (see Figure 5.33(a)). Obviously, powerful ECCs were necessary to deliver the packets successfully. One observes that the most retransmissions were encoded with BCH(63,39). Moreover, most packets that were sent using BCH(63,39) were retransmissions. One may now insist and ask why the adaptive FEC mechanism in such a case did not switch to the next powerful ECC: BCH(63,36). A possible explanation is that SSRA-FEC is slow in adapting to higher and more powerful codecs. It is possible that the interferences were frequent but not dense enough in time. Therefore, it is possible that this adaptive FEC mechanism did not

select more powerful ECCs, although it would have been better. The selection of the BCH(63,39) should have been preferred to the BCH(63,45), regarding the high percentage of retransmissions (green column) that were successfully delivered compared to the percentage of the successful original packet delivery (orange column).

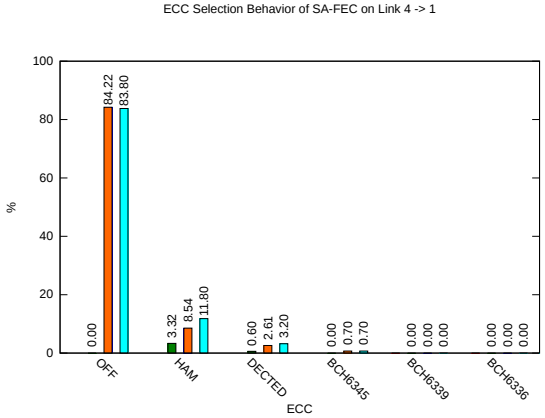
From a general point of view, to increase the PDR values and reduce the retransmissions, the SA-FEC wins the race. Compared to the other adaptive FEC approaches, SA-FEC reacts most quickly to link quality changes because it does not depend on a history such as the other two approaches. In addition to that, the histograms in Figures 5.31, 5.32 and 5.33 also clearly show that the A-FEC mechanisms work independently on every link, since there were quite different ECC selection behavior patterns on some links such as the link from msb1 to msb6. This per-link adaptation is one of the key advantages, since the most suitable ECC for a certain link is selected.



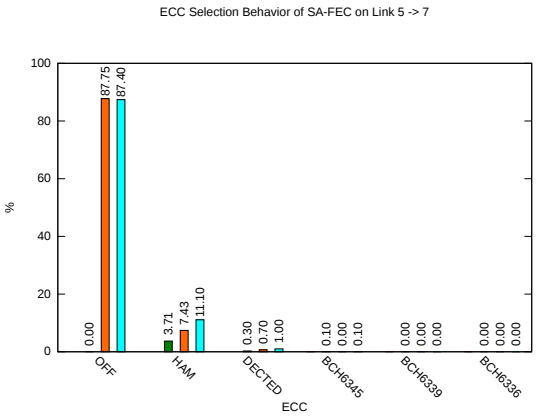

(a) Link 1:6



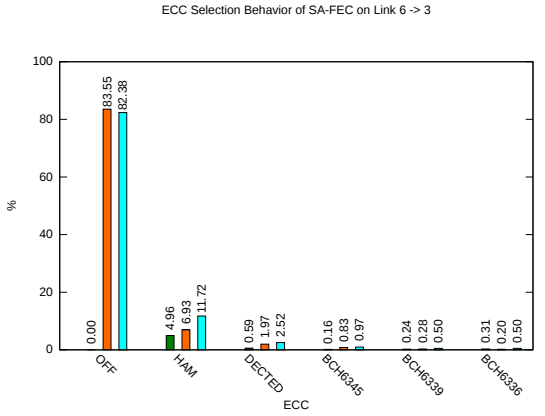

(b) Link 2:3



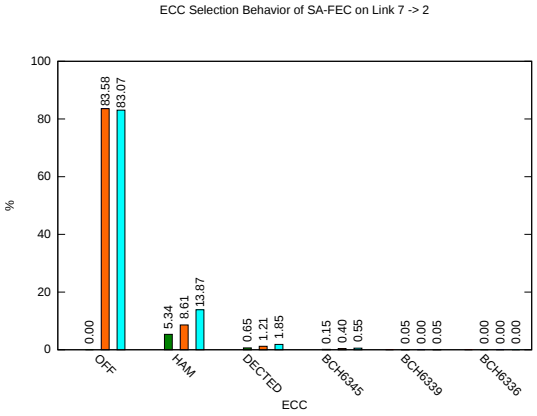

(c) Link 4:1



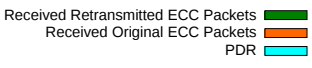

(d) Link 5:7




(e) Link 6:3

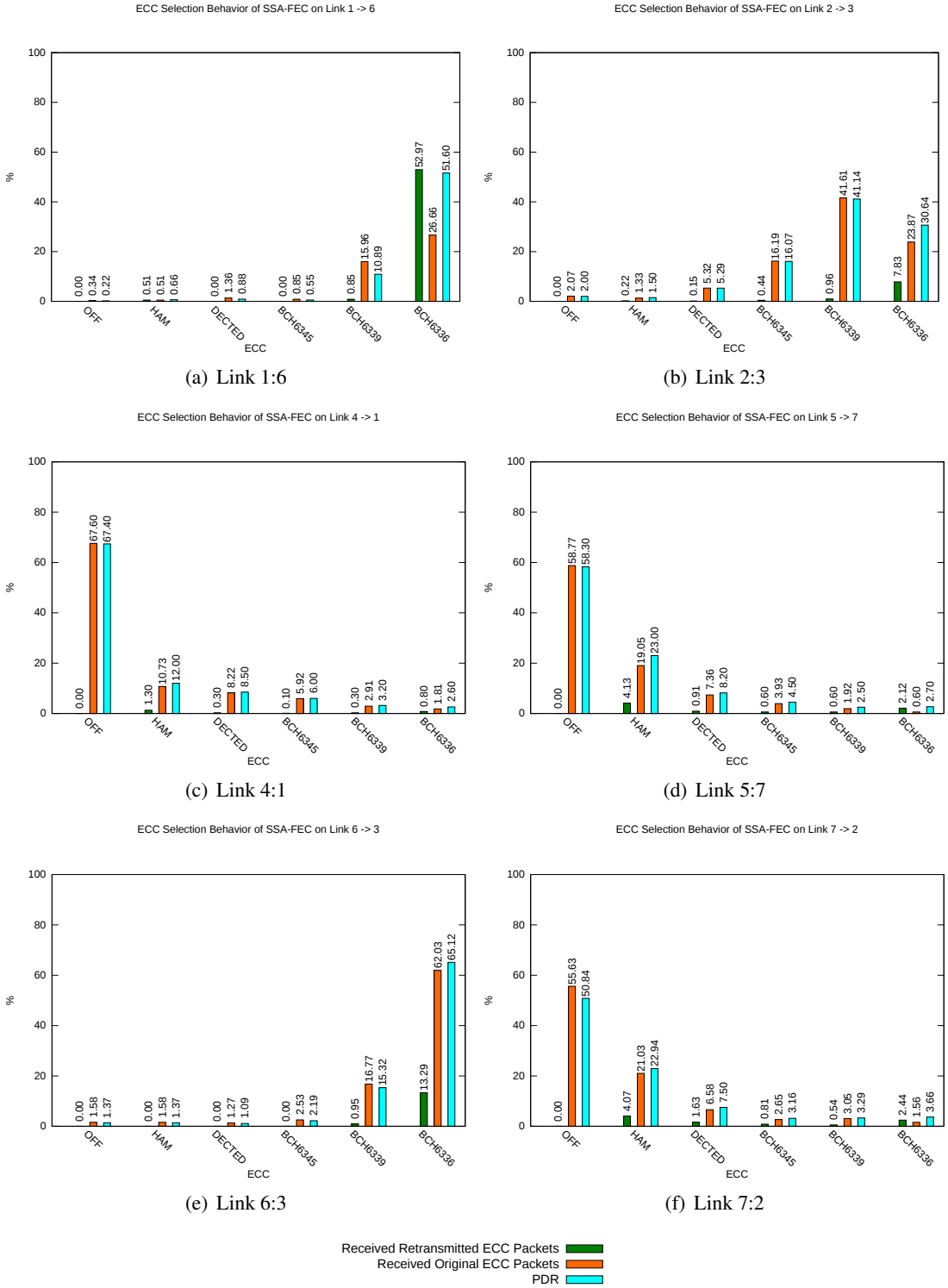



(f) Link 7:2




(g) Legend

**Figure 5.31:** ECC Selection Behavior of SA-FEC per Link




(a) Link 1:6

(b) Link 2:3

(c) Link 4:1

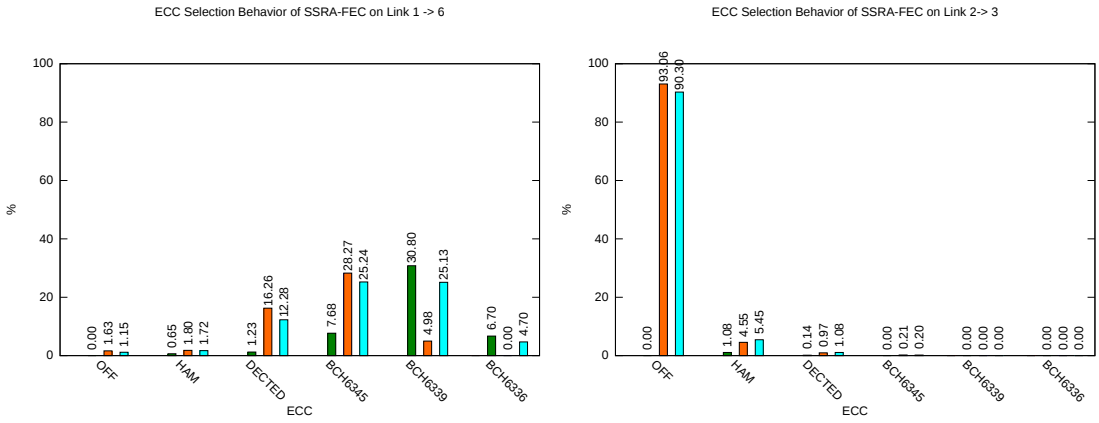
(d) Link 5:7

(e) Link 6:3

(f) Link 7:2

(g) Legend

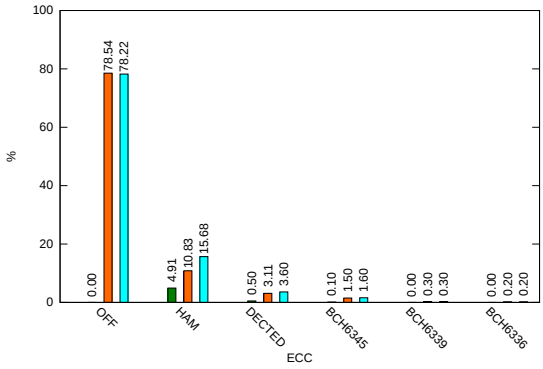
**Figure 5.32:** ECC Selection Behavior of SSA-FEC per Link



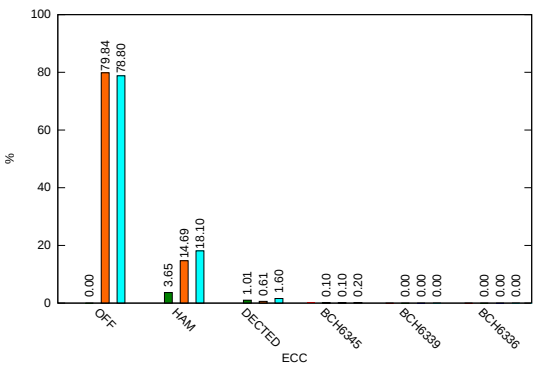

(a) Link 1:6



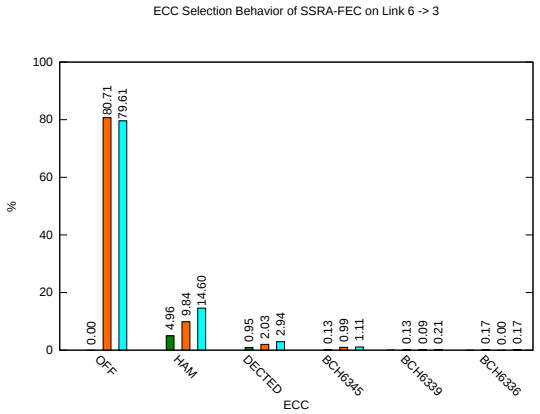

(b) Link 2:3



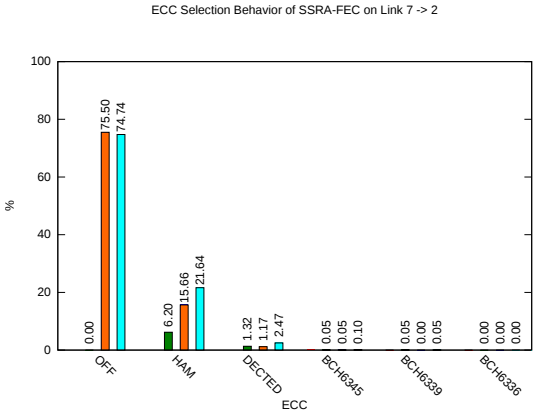

(c) Link 4:1



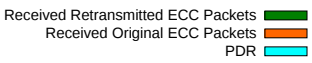

(d) Link 5:7




(e) Link 6:3




(f) Link 7:2




(g) Legend

**Figure 5.33:** ECC Selection Behavior of SSRA-FEC per Link

### Corrected Errors

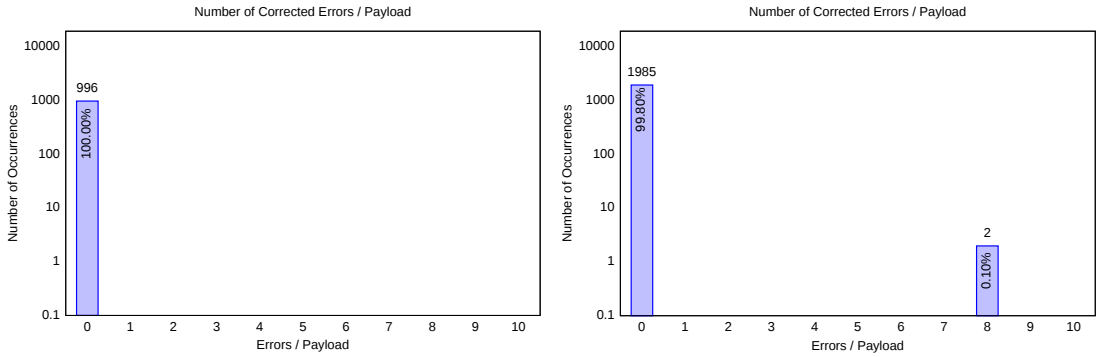
Figures 5.34, 5.35 and 5.36 depict the analysis of the corrected errors per payload for the A-FEC mechanisms. The data evaluated here, origins from the same run as the results above. The histogram for the SA-FEC shows that msb6 corrected significantly more errors than the other sensor nodes. This is not surprising, since we already noticed in Figure 5.29(b) that on the link from msb1 to msb6 the lowest PDR was measured compared to the other links. The low PDR on this link and the high number of corrected errors can be connected. They indicate that the interferences leading to bit errors on this link were present.

Generally, one observes the same trend that multiple bit errors per payload are infrequent in the histogram in Figure 5.34(d) as in the other corrected error measurements. If we relate the results of the corrected error analysis with the distribution of the packets among the ECCs as shown in Figure 5.31(a), we can see that SA-FEC adapted to the situation. On the other hand, for the other nodes mostly no ECC has been used, which relates to the fact that not many errors have been corrected on these nodes. Considering the results from the PDR measurements for the links using SA-FEC (see Figure 5.29(b)), it can be said that the switching between the ECCs worked surprisingly well. Obviously, SA-FEC made the suitable estimations which ECCs to apply for the next transmission.

In Figure 5.35, the corrected error statistics for the A-FEC approach SSA-FEC are depicted. Compared to the error results of SA-FEC (see Figure 5.34), the pattern is similar. The node msb6 corrected the most errors per payload. Also, the trend that more errors per payload are less frequent, which we described in the previous paragraph can be seen here. If we relate the corrected errors for msb6 and the distribution of the selected ECCs shown in Figure 5.32(a), the SSA-FEC needed much more powerful ECCs than the SA-FEC. As explained in Section 5.3.4, it is likely that not only the amount of errors in the ECC packet payload causes the switch to more powerful ECCs, but also the loss of ACK messages which are not FEC encoded and therefore, not protected at all. Unfortunately, if we would protect the ACK messages with FEC against bit errors, it is still not guaranteed that they are delivered successfully.

If we analyze the corrected errors of msb3 and relate it to the distribution of used ECCs in the Figures 5.32(b) and 5.32(a), a similar explanation can be given to explain the selection of powerful ECCs. Although no error is corrected at msb3, most packets are encoded with powerful ECCs. But the increased amount of retransmissions that are successfully received indicated that it is mainly the loss of ACK messages that let the SSA-FEC choose more powerful ECCs.

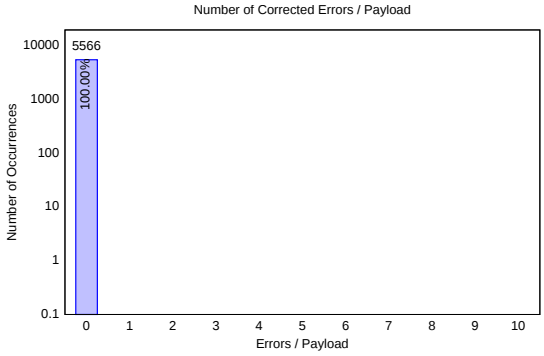
In Figure 5.36, the adaptive FEC approach SSRA-FEC draws a similar picture concerning corrected errors per payload as the other two adaptive FEC mechanisms. Therefore, it is not surprising, that msb6 did a lot of the correction work. This also reflects the distribution of the ECCs for this approach depicted in Figure 5.33. Only msb6 tended to use stronger ECCs. Therefore, more errors were corrected. This observation is valid for all the applied A-FEC mechanisms. We conclude that this is a environmental effect. Apparently, msb6 is located in a noisy environment. Relating the corrected error statistics of msb6 to the ECC selection behavior of the link from msb1 to msb6 (see Figures 5.31(a), 5.32(a) and 5.33(a)), it is not surprising that on this link the A-FEC mechanisms tend to use more powerful ECCs. As well as in the results from the static FEC error measurements described in Section 5.3.3, some isolated columns in the histograms occur, which however often remained statistically insignificant.



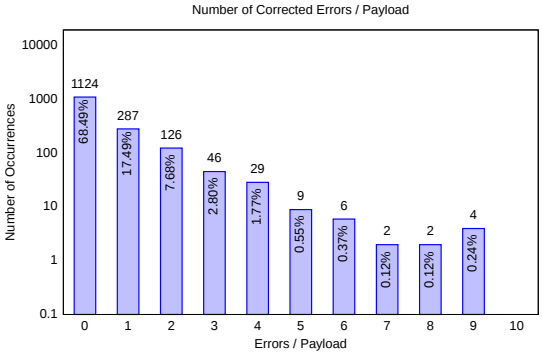

(a) msb1



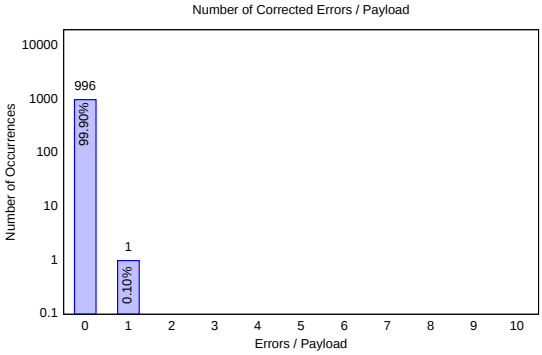

(b) msb2




(c) msb3

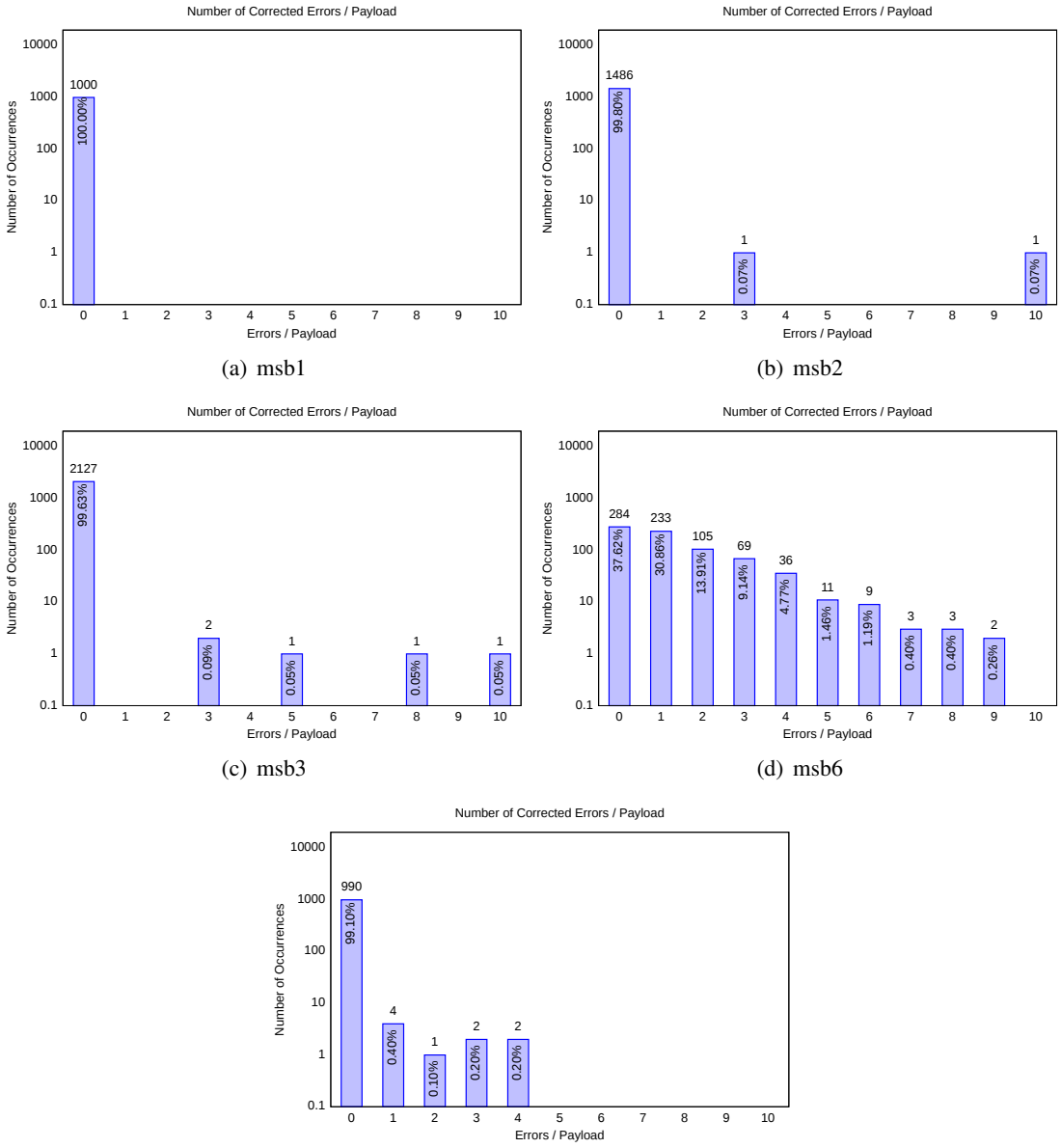



(d) msb6




(e) msb7

**Figure 5.34:** Corrected Errors SA-FEC




(a) msb1

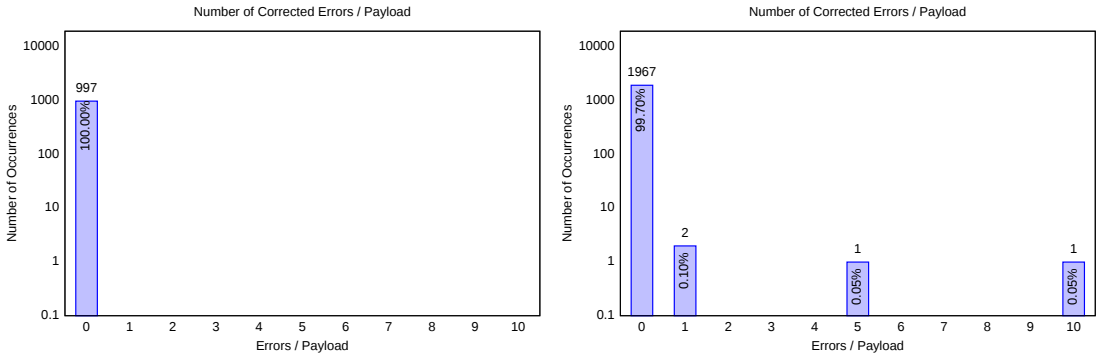
(b) msb2

(c) msb3

(d) msb6

(e) msb7

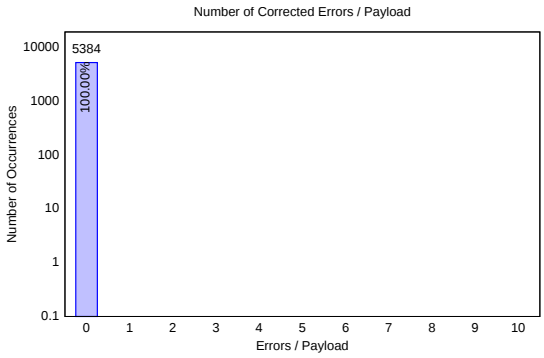
**Figure 5.35:** Corrected Errors SSA-FEC



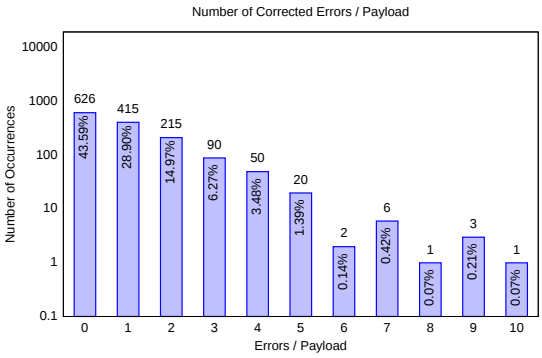

(a) msb1



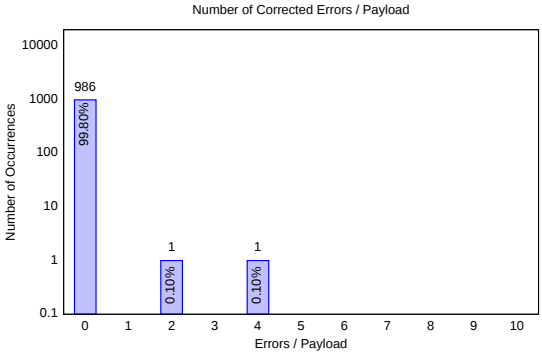

(b) msb2




(c) msb3




(d) msb6




(e) msb7

**Figure 5.36:** Corrected Errors SSRA-FEC

# Chapter 6

# Conclusions and Outlook

## 6.1 Conclusions

In this Master thesis, we implemented several ECCs into ScatterWeb OS on the ultra low-power MSB430 wireless sensor nodes. The purpose of ECCs is to cope with transmission errors. These errors appear as bit flips in the transmitted packets. On wireless links, such errors can occur due to a variety of reasons, for example interferences caused by reflection from obstacles or ongoing radio transmissions. Using no ECC, these bit errors lead to corrupted packets which the receiver drops and which have to be recovered through retransmissions. With the implementation of several ECCs, we showed that on one hand it is possible to use ECCs on the MSB430 sensor nodes and on the other hand that for low quality links the PDR can be increased using ECCs. Since the link quality can change suddenly, we developed three FEC mechanisms that react to such changes. The evaluation of the experiments in the multi-link topology illustrates that in most cases, Hamming(7,4) and DECTED(16,8) are sufficient to overcome the occurring errors. This observation is also congruent with the fact that most errors are one and two bit errors. Considering PDR, one can say that the adaptive approaches we implemented succeeded in significantly increasing the packet delivery reliability.

The results of the corrected errors also convey that errors are a local matter within a topology with multiple links. It happened that some ECCs corrected a lot more errors than others. From this point of view, the usage of our adaptive FEC approaches makes sense, because the adaptation takes place on a per-link basis. Moreover, if we analyze the time needed for encoding and decoding by the different ECCs, the A-FEC approaches benefit from the selection of the appropriate ECCs for a given link quality. Always using the most powerful ECC, which in our case is BCH(63,36) may achieve good results considering PDR, but uses a lot of time for encoding and decoding and requires the CPU to consume a lot of energy. Besides that, the results show that the most errors can be corrected with simpler ECCs. Using the A-FEC mechanisms, the most appropriate ECC according to the link quality is selected. The application of SA-FEC, SSA-FEC or SSRA-FEC does not need any knowledge of possible errors which could occur. The A-FEC mechanisms determine at runtime for every link, which ECC should be applied. As the results show, interferences which cause bit errors can be a local phenomenon affecting only single links and are very hard to predict. With applying our A-FEC mechanisms, the user does not need to be concerned about such phenomena, since the A-FEC will adapt to such a situation.

In an attempt to obtain results which are close to reality, we did not use any simulated topologies to run the experiments, since simulation experiments are inherently unreliable when dealing with phenomena which are as tightly linked to channel characteristics as FEC. The experiments ran in a wireless sensor node topology within a building and across two buildings. The advantage is that the influences on the topology are real and do not follow a simplified model, in contrary to a simulator, where for example an error model is specified by the user. This allows us to use the findings as a general conclusion for other real-world wireless sensor network applications. The drawback of such real-world experiments is that the reproduction of the results is very difficult.

Nevertheless, we can conclude that with respect to the reliability of a transmission, the application of any ECC is an achievement.

## 6.2 Outlook

A further point that could be investigated is the measurement of the energy consumption of the implementation, as done by Hurni et al. [38]. Generally, in wireless sensor networks the energy constraints are a big issue. Therefore, it makes sense to classify the ECCs based on their energy cost. We only estimated the energy costs for the CPU for the encoding and decoding process of the implemented ECCs (see Section 5.2). To quantify the total energy costs, measurements of the energy consumption of the entire sensor node should be done. Further, the energy costs of the ECCs in combination with a energy-efficient MAC protocol is a topic to investigate. The standard implementation of the ScatterWeb OS does not use an energy-efficient MAC protocol. This means that the transceiver is always on and does not save energy through powering off. Since the transceiver needs a significant amount of energy during operation, without an energy-efficient MAC protocol, the measurement of the energy costs would not make sense. Since the integration of an energy-efficient MAC protocol is beyond the scope of this thesis, the energy consumption has not been directly measured, but only estimated.

In our implementation only the payload of the packet is protected against bit errors. Since bit errors in header fields can harm a packet severely, one might extend the protection through an ECC from the payload to some important header fields, such as the packet size or the packet type. Nevertheless, it is not possible to protect all header fields with ECC. For example, the receiver node ID should be left not encoded since otherwise, every node receiving the packet needs to decode the receiver address field first to check if the packet is really designated for it. This would lead to an unnecessary waste of resources on nodes which hear the packet, but are not intended to receive the packet. On the other hand, bit errors in the receiver address field are severe, since the receiver does not notice that there is a packet sent to it.

We did not investigate burst errors. Such errors affect several continuous bits in the same code word. As soon as multiple bits per code word are corrupted, more powerful and resource consuming ECCs are required. In order to cope with this kind of bit errors, one could apply code word interleaving. This means that burst errors are distributed over several code words and therefore less powerful ECCs are able to correct the errors. This would introduce an additional overhead to manage the code words, which may reduce the benefit of applying code word interleaving or even are too resource consuming for the MSB430 sensor nodes.

As already mentioned in Section 2.1, one could also compare the A-FEC mechanisms as well as the static ECCs with ARQ. ARQ can be seen as a standalone mechanism or as an option which can be applied to the implemented FEC mechanisms. This would lead to a huge amount of additional measurement scenarios that would exceed the time frame of this thesis. Moreover, pure ARQ does not provide any bit error correction capabilities, and is therefore not within the scope of this thesis.

# Bibliography


[1] Wikipedia, "File:hamming(7,4).svg — wikipedia, the free encyclopedia," 2006, [Online; accessed 4-January-2011]. [Online]. Available: http://en.wikipedia.org/wiki/File:Hamming%287,4%29.svg

[2] M. Baar, A. Liers, and J. Schiller, "Poster abstract: The scatterweb msb-430 platform for wireless sensor networks," 2007.

[3] ScatterWeb Homepage. [accessed 21-Dezember-2010]. [Online]. Available: http://cst.mi.fu-berlin.de/projects/ScatterWeb/mod_MSB-430.html

[4] "Cc1020 single chip low power rf transceiver for narrowband systems," [Online; accessed 15-April-2011]. [Online]. Available: http://www.tkn.tu-berlin.de/curricula/ss06/pr/CC1020_Data_Sheet_1_6.pdf

[5] G. Fairhurst and L. Wood, "Advice to link designers on link Automatic Repeat reQuest (ARQ)," RFC 3366 (Best Current Practice), Internet Engineering Task Force, Aug. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3366.txt

[6] T. Braun, M. Diaz, J. Enrquez-Gabeiras, and T. Staub, *End-to-End Quality of Service Over Heterogeneous Networks*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[7] W. Peterson and D. Brown, "Cyclic codes for error detection," *Proceedings of the Institute of Electrical and Electronic Engineers (IRE)*, vol. 49, no. 1, pp. 228 –235, January 1961.

[8] C. E. Shannon, "A Mathematical Theory of Communication," *SIGMOBILE Mobile Computer Commununications*, vol. 5, pp. 3–55, January 2001.

[9] K. Beuth, *Elektronik 4. Digitaltechnik*, ser. Vogel Fachbuch. Vogel Buchverlag, 2003.

[10] S. Lin and J. D. J. Costello, *Error Control Coding: Fundamentals and Applications second edition*. Prentice Hall: Englewood Cliffs, NJ, 2004.

[11] R.H. Morelos-Zaragoza, *The Art of Error Correcting Coding*. John Wiley, 2006. [Online]. Available: http://books.google.com/books?id=yIgZAQAAIAAJ

[12] R. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, 1950.

[13] Wikipedia, "Majority logic decoding — wikipedia, the free encyclopedia," 2010, [Online; accessed 4-January-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Majority_logic_decoding&oldid=343875110

[14] S. Ling and C. Xing, *Coding theory: a first course*. Cambridge University Press, 2004. [Online]. Available: http://books.google.com/books?id=N1jiL8V3ISwC

[15] T. Moon, *Error correction coding: mathematical methods and algorithms*. Wiley-Interscience, 2005. [Online]. Available: http://books.google.ch/books?id=X6TADZB7vokC

[16] T. A. Gulliver and V. K. Bhargava, "A systematic (16,8) code for correcting double errors and detecting triple-adjacent errors," *IEEE Transactions on Computers*, vol. 42, January 1993.

[17] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres (Paris) 2*, pp. 147–156, September 1959.

[18] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, no. 1, pp. 68 – 79, 1960.

[19] E. R. Berlekamp, *Algebraic Coding Theory*. McGraw-Hill, 1968.

[20] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, January 2003.

[21] R. T. Chien, "Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes," *IEEE Transactions on Information Theory*, vol. 10, no. 4, pp. 357–363, 1964.

[22] J. Hong and M. Vetterli, "Simple algorithms for BCH decoding," *IEEE Transactions on Communications*, vol. 43, no. 8, pp. 2324–2333, 1995.

[23] J. Jeong and C. T. Ee, "Forward error correction in sensor networks." International Workshop on Wireless Sensor Networks (WWSN), 2007.

[24] A. Willig and R. Mitschke, "Results of bit error measurements with sensor nodes and casuistic consequences for design of energy-efficient error control schemes," in *European Workshop on Sensor Networks (EWSN)*, 2006, pp. 310–325.

[25] T. K. Marcel Busse, Thomas Haenselmann and W. Effelsberg, "The Impact of Forward Error Correction on Wireless Sensor Network Performance." ACM Workshop on Real-World Wireless Sensor Networks (REALWSN), 2006.

[26] J.-S. Ahn and J. Heidemann, "An adpative FEC algorithm for mobile wireless networks," USC/Information Sciences Institute, Tech. Rep. ISI-TR-555, March 2002. [Online]. Available: http://www.isi.edu/~johnh/PAPERS/Ahn02a.html

[27] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET simulation studies: the incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, 2005.

[28] T. R. Andel and A. Yasinsac, "On the Credibility of Manet Simulations," *IEEE Computer Magazine*, 2006.

[29] M. Baar, E. Koeppe, A. Liers, and J. Schiller, "The scatterweb msb-430 platform for wireless sensor networks." SICS Contiki Workshop, 2007.

[30] Freie Universitaet Berlin. ScatterWeb$^2$ OS. [accessed 21-Dezember-2010]. [Online]. Available: http://scatterweb.mi.fu-berlin.de

[31] C. Liechti and D. Diky, "The gcc toolchain for the texas instruments msp430 mcus," 2011, [Online; accessed 17-March-2011]. [Online]. Available: http://mspgcc.sourceforge.net/

[32] Willow, "Telos-b," 2011, [Online; accessed 17-March-2011]. [Online]. Available: http://www.willow.co.uk/TelosB_Datasheet.pdf

[33] INTECH, "Kmote-b," 2011, [Online; accessed 17-March-2011]. [Online]. Available: http://www.tinyosmall.com/product_p/100-101.htm

[34] "Cyclic redundnacy checks," [Online; accessed 14-April-2011]. [Online]. Available: http://www.mathpages.com/home/kmath458.htm

[35] D. Puccinelli, O. Gnawali, S. Yoon, S. Santini, U. M. Colesanti, S. Giordano, and L. J. Guibas, "The impact of network topology on collection performance." European Conference on Wireless Sensor Networks (EWSN), 2011.

[36] 16-Bit Ultra-Low Power MSP430 Microcontrollers, "Datasheets and Reference Manuals," http://focus.ti.com/.

[37] A. Hergenröder, J. Wilke, and D. Meier, "Distributed Energy Measurements in WSN Testbeds with a Sensor Node Management Device (SNMD)," in *International Conference on Architecture of Computing Systems*, 2010.

[38] P. Hurni, B. Nyffenegger, T. Braun, A. Hergenroeder, "On The Accuracy of Software-based Energy Estimation Techniques." European Conference on Wireless Sensor Networks (EWSN), 2011.

[39] Buettner, M., Gary V. Y., Anderson, E. and Han, R., "X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks." ACM SenSys, 2006.

[40] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks." ACM SenSys, 2004.

[41] P. Hurni, G.Wagenknecht, M. Anwander, and T. Braun, "A Testbed Management System for Wireless Sensor Network Testbeds (TARWIS)." European Conference on Wireless Sensor Networks (EWSN), February 17-19, Coimbra, Portugal, 2010.