REIHE INFORMATIK
TR-2006-005

The Impact of Forward Error Correction on Wireless Sensor Network Performance

Marcel Busse, Thomas Haenselmann,
and Wolfgang Effelsberg

Universität Mannheim
Praktische Informatik IV
Seminargebäude A5
D-68159 Mannheim, Germany

The Impact of Forward Error Correction on Wireless Sensor Network Performance

Marcel Busse, Thomas Haenselmann, and Wolfgang Effelsberg
Computer Science IV, University of Mannheim, Seminargebäude A5
D-68159 Mannheim, Germany
{busse, haenselmann, effelsberg}@informatik.uni-mannheim.de

***Abstract*—In networks there are basically two methods to tackle the problem of erroneous packets: Automatic Repeat Requests (ARQ) and Forward Error Correction (FEC). While ARQ means packet retransmissions, FEC uses additional bits to detect and correct distorted data.**

However, extensive field test of our sensor nodes have shown that FEC can take effect only as long as both sender and receiver are bit-wise synchronized. Otherwise, all following bits are misinterpreted which results in an uncorrectable number of errors. We will thus introduce a new resync scheme which is particularly tailored for many sensor network platforms using UARTs in conjunction with radio transmission. We can show that only using an appropriate resync mechanism exploits the full potential of FEC.

I. Introduction

In traditional 802.x based WLANs, the strength of a radio signal will usually be sufficient to reliably cover a relevant area. If not, a larger number of base stations will be used. Thus, many evaluations come to the conclusion that incorrect packets are distorted by burst errors in wireless LANs caused by disturbing signal sources. Here, error recovery by retransmission is more efficient than adding redundant information to every packet in advance.

However, Riemann and Winstein have shown that outdoor settings and radio communication in a line of sight over large distances or with weak radio signals exhibit different error characteristics [1]. Single bit errors occur with a much lower variance which means that they are more evenly distributed.

For example, consider a large packet with a size of 256 bytes. If we assume that there is a 50% chance for a single bit error within 500 bits, the likelihood of receiving an error free packet of 256 bytes is as low as 6%. Despite the small number of single bit errors we would require to retransmit more than 16 packets on average in order to get one through. On the other hand, the small number of bit errors could be compensated easily by *Forward Error Correction* (FEC).

However, as we will see in the next section, in most cases erroneous packets are completely destroyed because the sender and receiver got out of sync during the radio transmission. Thus, rather ARQ than FEC is advisable [2]. But if both sender and receiver are resynchronized periodically, FEC will be more efficient. We will describe such a resync mechanism in the next section. Section III then reviews some FEC codes that are evaluated in Section IV. Finally, Section V concludes the paper.

II. Resync

We have shown in [3] that a significant amount of errors are introduced by the ESB hardware itself. These errors are due to some common hardware components being used, particularly the so-called *Universal Asynchronous Receiver-Transmitter* (UART) that interconnects the radio transceiver and the CPU. UARTs are used to enable the asynchronous communication by generating start-, data- and stop-bits. Start-bits indicate the beginning of a new byte, and the UART waits in an idle state until a start-bit is detected. Thus, the missing of a start-bit might cause the receiver's UART state machine to get out of sync. More precisely, all following bits are shifted to the left with regard to the true bit-stream being sent. Getting the stream re-synchronized is the task of our resync mechanism.

In order to re-synchronize the sender and the receiver state machines, the sender sends eight high values (called the *resync byte*) followed by a stop-bit. Since start-bits are indicated by low values, the receiver will remain in a state "waiting for a new start-bit", no matter how many bits the receiver fell behind before. Then with the next byte, both state machines are bit-synchronized again.

In order to achieve a resync at the byte level, the following approach is used: Let n be the number of bytes after a resync byte is stuffed into the stream and m be the number of yet received data bytes by the receiver. The stuffing of resync bytes starts with the first data byte, thus the receiver expects one each n bytes, i.e., if

$$m \mod n = 0. \quad (1)$$

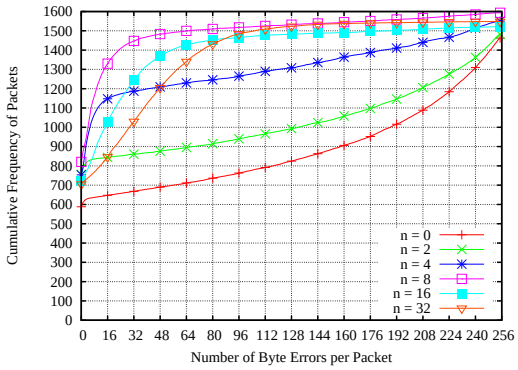

Fig. 1. Cumulative Distribution of Byte Errors per Packet [3]

When a resync byte is detected, the receiver verifies if the byte stream is still synchronous. Otherwise, Equation 1 is false and so is the receivers byte position. Since it is likely that the receiver is some bytes behind, it increases its byte position until Equation 1 is satisfied again.

If no resync byte could be recognized but m becomes a multiple of n, the receiver might have missed the resync byte. However, it is likely that the resync byte was considered as a data byte and therefore missed. Hence, the current byte is skipped by the receiver.

The frequency with which resync bytes should be stuffed into the data stream was also analyzed in [3]. Figure 1 shows the cumulative frequency vs. the number of byte errors per packet[1]. For $n \to \infty$, resync bytes are used sparsely so that they might be effectless. On the other hand, for $n \to 1$, resync bytes are used too often so that the byte stream will be *DC unbalanced*.

As it is shown in Figure 1, the best trade-off is achieved for $n = 8$ with the highest number of correctly received packets. If no FEC would be used, the number of packets containing no error is improved by about 35%. If up to 16 or 32 errors per packet are allowed, the number of packets is even more than twice as high.

The significant increase in the delivery rate with up to e errors motivates us to further analyze different FEC schemes that are able to correct such errors. If no resync is used at all, i.e., for $n = 0$, FEC would improve the delivery rate only marginally since most of the erroneous packets are completely destroyed. This was also recently reported in [2]. However, the resync mechanism now enables the usage of FEC.

III. Forward Error Correction

In this section, we will review three FEC codes that might be able to correct certain kinds of errors. A comprehensive overview of FEC schemes, also in comparison with *Automatic Repeat Request* (ARQ) techniques, can be found in [4].

[1]The evaluation settings were very similar to the setup used in Section IV.

A. Linear Block Codes

Let $\mathbb{B} = \{0, 1\}$ and $n \in \mathbb{N}$. Then a code $C \subseteq \mathbb{B}^n$ is called a *linear block code* if the sum of two codewords u and v is a codeword, too. Let $k \in \mathbb{N}$ be the dimension of C with $\{b_1, \ldots, b_k\}$ being a basis of C. The $k \times n$ matrix

$$G = [b_1 \ldots b_k]^T$$

is called the *generator matrix* of the code C since C will be constructed by G with

$$C = \{xG \mid x \in \mathbb{B}^k\}.$$

Thus, C is also called a (n, k) code where k refers to the number of data bits and n refers to the number of bits the encoded codeword will have.

With I_k be the $k \times k$ identity matrix, G can be transformed to $[I_k P]$, where P is a $k \times r$ binary matrix and r refers to the number of *parity bits* that are added to the code C. Using the generator matrix G, the *parity matrix* H is constructed and is of the form $[P^T I_r]$. Then, for all codewords $x \in \mathbb{B}^n$ we get

$$x \in C \quad \Leftrightarrow \quad xH^T = 0 \tag{2}$$

where xH^T is called the *syndrome* s of x. If $s \neq 0$, an error occurred during the encoding process.

B. Hamming Codes

Hamming codes are linear (n, k) codes that are able to correct 1-bit errors [5]. The parity matrix H can be constructed in a simple way by setting the j-th column vector to the binary form of number j. Note that then H and G are not in the form $[P^T I_r]$ resp. $[I_k P]$, but can easily be transformed to it by exchanging some column vectors. If a one-bit error occurs, the syndrome s will give the position of the erroneous bit. Let $c \in \mathbb{B}_n$ be a valid codeword that is received as $x = c + e$ with a 1-bit error vector $e \in \mathbb{B}_n$. Then we get

$$s = xH^T = (c + e)H^T = eH^T.$$

Since eH^T corresponds to the e-th column in H, s gives the position of the error.

As the Hamming code is able to correct 1-bit and detect up to 2-bit errors, it is referred to as a *single-error-correcting* and *double-error-detecting* (SEC-DEC) code. Other SEC-DEC codes are for example the *odd-weight-column* codes [6] where each column of H has an odd *weight*, i.e., the number of ones is odd. If the number of ones is furthermore balanced in each row, the code can be implemented very efficient [7].

C. Double Error Detection Codes

A *double-error-correcting* and *triple-error-detecting* (DEC-TED) $(16, 8)$ code was proposed by Gulliver and Bhargava. Please refer to [8] for the definitions of H and G. If a 1-bit or 2-bit error occurs, the syndrome s will either be equal to a single column of H or to an additive (exclusive-or) combination of two columns. The index of the involved columns then represents the position of the bit error(s).

D. Interleaving

In order to combat the effects of burst errors, *interleaving* can be used: Two or up to k codewords are interleaved before they are transmitted. The number of interleaved codewords refers to the *depth* of an interleaver. The interleaver first stores the k codewords of size m in a $k \times m$ buffer row-by-row. We thus call such an interleaver a (k, m)-interleaver. It then outputs the interleaved codewords column-by-column. Thus, in the output stream there are always $k - 1$ other bits between two successive codeword bits. At the other end, a deinterleaver works in the reverse way.

If the interleaving depth is sufficiently large, the correlation between two successive codeword bits will be minimized. The deinterleaver might thus have enough capability to decode the codeword successfully. As it is shown in [9], interleaving is effective if tk exceeds the average burst length, where t denotes the number of errors the code is able to correct. For example, consider a stream of 16 bytes where the first two bytes are completely disturbed. A $(8, 8)$-interleaver would produce an output stream with 16 bytes where 8 bytes contain a two-bit error. On the other hand, a $(16, 8)$-interleaver produces an output stream with 16 bytes where all bytes contain only a one-bit error.

E. Reed Solomon Codes

Reed Solomon (RS) codes [10] are widely used, including digital television, wireless and satellite communication, broadband modems, CD's and DVD's, etc. Like the abovementioned FEC codes, they work by adding some redundancy to the original data that is later used to correct a certain amount of errors. However, RS codes do not correct several single bit errors but complete *symbols*, which contain a fixed number of bits. As before, the number of errors being correctable mainly depends on the amount of information added. For further information on the encoding and decoding algorithm, please refer to [11] or [12].

RS codes belong to the class of *systematic linear block codes*. Systematic means that the encoded data consists of the original data and some redundant symbols added to the end (called the *code block*). Each block is divided into several m-bit symbols with a symbol size of typically 3 to 8 bits. Finally, the linearity property of the code ensures that every possible m-bit word is valid for encoding.

Then, $RS(n, k)$ specifies a RS code with n encoded m-bit symbols per code block. The number of original symbols is specified by k, thus $n - k$ refers to the amount of redundancy used. With $2t = n - k$, a RS decoder will be able to correct up to t symbol errors. That means that either only t bits (single bit errors) or up to tm bits (all symbol bits erroneous) may be corrupted, depending on how many symbols are affected. Therefore, RS codes are able to correct burst errors of up to t unknown symbols.

If the position of a symbol error is known, such an error is called an *erasure*. Erasures are easier to correct, thus a codeword containing r unknown symbol errors and s erasures will still be recovered correctly if $2r + s \leq 2t$.

IV. Evaluation

We will evaluate three FEC codes that were discussed in the last section: The Hamming $(12, 8)$ code, the DEC-TED $(16, 8)$ code, and the $RS(255, 223)$ Reed-Solomon code. The evaluation setup will be discussed in the next section. Due to the RS code, the packet data size was set to 223 bytes for all FEC codes. Thus, the redundancy in bytes added to a packet is 112 bytes for the Hamming code, 223 bytes for the DEC-TED code, and 32 bytes for the RS code. However, note that the Hamming and the DEC-TED code are able to correct one-bit resp. two-bit errors per byte while the RS code is based on 8-bit symbols and will thus correct up to 16 unknown symbol errors.

A. Setup

We placed 16 ESB nodes (including one *source* node) in line on our office floor, each at a distance of 1 m as depicted in Figure 2. The source node was powered by a power supply unit while all other nodes used rechargeable batteries. Thus, we hope to get similar signal strengths for outgoing packets during each evaluation run.

In order to get different reception characteristics, the transmission power is varied from 100 down to 0 in step sizes of four. For each transmission power, the source node broadcasts 100 data packets containing 255 bytes[2]. The data rate was set to 1 packet/sec. Nodes receiving a data packet do not send an acknowledgment but quietly log the packet in a receiving buffer. Once the source node broadcast its data packet, it polls each node to get information concerning the packet reception. The polling is

[2]Note that no packet forwarding or routing are employed.

Fig. 2. Placement of 16 ESB Nodes on our Office Floor

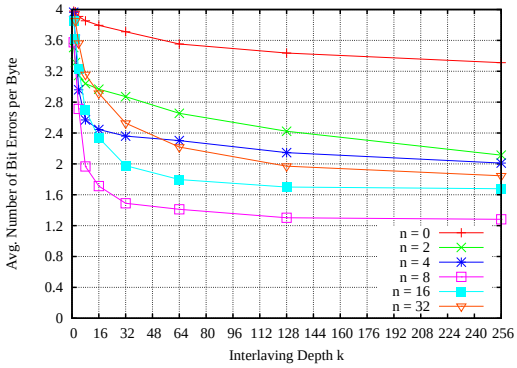

Fig. 3. Bit Errors per Byte for Increasing Interleaving Depths

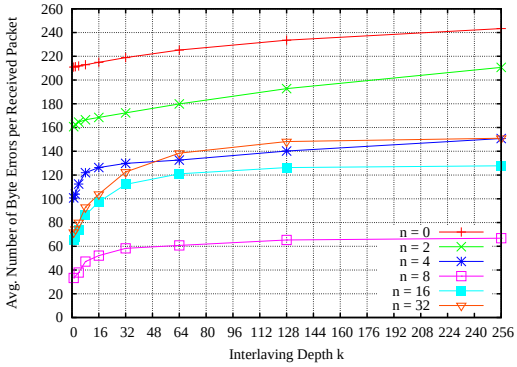

Fig. 4. Byte Errors per Received Packet for Increasing Interleaving Depths

performed with the maximum transmission power and repeated until all information are received correctly. Nodes that are polled by the source node answer with the data currently contained in the receiving buffer. Due to the limited node's memory space, the source node transmits the polled information to a notebook over a serial connection. The notebook then logs all received data in a database for later processing.

This setup is repeated for different resync frequencies n since the number of resyncs per packets influences the reception characteristics remarkably. Based on the log files we then evaluated the three FEC codes. As we will see in the next section, the performance will be improved if the data is interleaved as described in Section III-D. Thus, we will also investigate the impact of different interleaving depths k.

B. Measurements

We first analyze the number of bit errors that occur per erroneous byte. Figure 3 depicts the number of errors for different resync frequencies n and increasing interleaving depths k. As it is shown, the influence of resync is actually very small if no interleaving is performed since occurring errors are not spread over the packet. For this case, the number of bit errors per erroneous byte is four which is equal to the mathematical expectation. If the interleaving depth increases, the number of bit errors can be minimized, in the best case down to 1.2 bit errors on average for a resync frequency of $n = 8$. This is in accordance with [3] where we have shown that a resync frequency of 8 trades off the number of resyncs and byte errors per packet best (see Figure 1).

As shown in Figure 4, a packet contains about 210 byte errors on average without resync. For $n = 8$ the errors per packet are reduced to about 35. However, for resync frequencies above 8 it becomes worse again since the resync is performed too rarely.

Another effect that is shown in Figure 4 is that the number of byte errors increases with an increasing interleaving depth. Since interleaving is not able to reduce the total number of bit errors per packet but only spreads burst errors, the number of bytes containing at least one bit error increases if the number of interleave bytes increases, too.

At this point we can already conclude that interleaving might be beneficial for the SEC-DED$(12, 8)$ and DEC-TED$(16, 8)$ codes since the number of bit errors per byte are reduced with an increasing interleaving depth, but it will certainly affect the RS$(255, 223)$ code adversely because interleaving increases the total number of erroneous bytes being affected.

How the number of bit errors per byte are distributed is depicted in Figure 5 for a resync frequency of $n = 8$. Without interleaving, about 90% of erroneous bytes contain at least one bit error, 80% at least two bit errors. Thus, these bytes would be uncorrectable for a SEC-DED code. For $k \to \infty$, the percentage of bytes containing at least two bit errors can be reduced to about 27%. However, note that the SEC-DEC code only allows at most one bit error among *12 bits* resp. two bit errors among *16 bits* for the DEC-TED code. Concerning the actual codeword length, the percentage of at least e bit errors would therefore be lower than the ones depicted in Figure 5. In addition, note

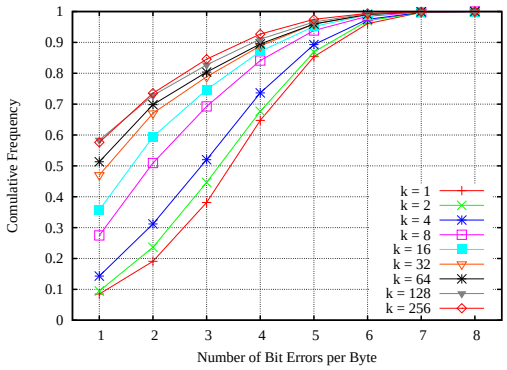

Fig. 5. Cumulative Distribution of Bit Errors per Byte

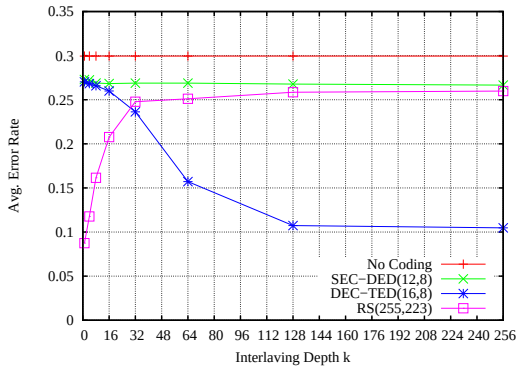

Fig. 6. Error Rate for Different FEC Codes

that any codeword within a packet must be correctable in order to decode the complete packet correctly.

Finally, Figure 6 depicts the error rate, which is defined as the fraction of received packets containing no errors, for all FEC codes using a resync frequency of $n = 8$. It also shows the error rate if no coding is used at all. The RS$(255, 223)$ code performs best, with about 8% of packets containing no errors, followed by the DEC-TED code with 11% and the SEC-DED code with 27%. The SEC-DED code only improves the error rate slightly compared to the case if no coding is used, even if the interleaving depth increases. Thus, it is likely that most packets contain at least one codeword that has more than one bit error, preventing the entire packet from being corrected.

Since interleaving is not able to reduce the number of bit errors on its own, it has no effect on the error rate if no coding is used. However, burst errors are spread very effectively, and in doing so interleaving reduces the maximum number of bit errors per byte on average but at the same time increases the number of affected bytes containing bit errors. Therefore, interleaving has a negative effect on the RS code. On the other hand, the DEC-TED code benefits from an increasing interleaving depth significantly and almost achieves the same error rate as the RS code. However, concerning the redundant bytes added to the data, the RS code requires only 32 additional bytes, the SEC-DEC code 112 bytes, and the DEC-TED code even 223 bytes.

V. Conclusion

We have shown significant improvements of the packet error rate by FEC, in case both the sender and receiver perform a byte-wise resync periodically. While the RS code is very efficient in terms of redundant bytes that are added to a packet, it is quite complex and requires much memory and processing time. The DEC-TED $(16, 8)$ code is easier to implement but requires twice as many bytes. However, both FEC codes achieve nearly the same error rates, if the DEC-TED code is combined with interleaving. The choice of an appropriate FEC code thus depends on the application the Sensor Network is used for.

References

[1] R. Riemann and K. Winstein, "Improving 802.11 range with forward error correction," Tech. Rep. MIT-CSAIL-TR-2005-011, Massachusetts Institute of Technologie, Cambridge (MA), USA, 2005.

[2] 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 *Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN)*, Zurich, Switzerland, February 2006.

[3] M. Busse, H. Haenselmann, and W. Effelsberg, "The Impact of Resync on Wireless Sensor Network Performance," Tech. Rep. TR-2006-004, University of Mannheim, Germany, 2006 (under submission).

[4] H. Liu, H. Ma, M. E. Zarki, and S. Gupta, "Error Control Schemes for Networks: An Overview," *Mobile Networks and Applications*, vol. 2, no. 2, pp. 167–182, 1997.

[5] R. W. Hamming, "Error Detection and Error Correction Codes," *The Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, 1950.

[6] M. Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395–401, 1970.

[7] S. Ghosh, S. Basu, and N. A. Touba, "Reducing Power Consumption in Memory ECC Checkers," in *Proceedings of the 4th IEEE International Test Conference (ITC)*, Charlotte, NC, USA, October 2004.

[8] 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, no. 1, pp. 109–112, January 1993.

[9] M. Zorzi and R. R. Rao, "Capture and Retransmission Control in Mobile Radio," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 8, pp. 1289–1298, 1994.

[10] I. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *SIAM Journal on Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[11] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.

[12] W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, MIT Press, 1972.