# Energy efficiency of error correcting mechanisms for wireless communication

*Paul J.M. Havinga*

University of Twente
department of Computer Science
the Netherlands
havinga@cs.utwente.nl

*Abstract*

We consider the energy efficiency of error control mechanisms for wireless communication. Since high error rates are inevitable to the wireless environment, *energy efficient error control* is an important issue for mobile computing systems. Although good designed retransmission schemes can be optimal with respect to energy efficiency, they can introduce problems to fulfill the required QoS of an application. Since the channel conditions and the requirements vary dynamically, we concentrate on *adaptive* implementations of *error correction* mechanisms.

Error correction mechanisms traditionally optimize the performance and provide an optimal solution to withstand all possible errors. In the context of energy consumption we have studied the energy efficiency of three different error correction mechanisms and have implemented these in software. We show that the energy efficiency trade-off between error correction and communication can have a great impact on the energy efficiency of a system. It will be shown that from energy efficiency perspective it is not always profitable to minimize the number of bits transmitted over the air.

A general strategy will be provided that can be used to determine an energy efficient error correction scheme of a minimal *system* consisting of a general-purpose processor and a wireless interface. As an example we have investigated a system with a WaveLAN card as wireless interface.

*Keywords*: error correction, Reed-Solomon, EVENODD, energy efficiency, wireless communication, Quality of Service

## 1. Introduction

Portable computers like PDAs and laptops that use wireless communication to interact with the environment must rely on their limited battery energy for their operation. Power consumption is becoming the limiting factor in the amount of functionality that can be placed in these devices [Havinga 97]. More extensive and continuous use of network services will only aggravate this problem [Stemm 96]. The design of the wireless communication protocols must be done very carefully and consider judicious use of the available energy. Since high error rates are inevitable to the wireless environment, *energy efficient error control* is an important issue for mobile computing systems. In this paper we investigate the energy efficiency of error control mechanisms.

Error control mechanisms traditionally trades off complexity and buffering requirements for throughput and delay. In classic error control schemes the main design criteria is on optimizing the performance and provide an optimal solution to withstand all possible errors [Lin 84, Liu 96, Cho 94]. In the context of energy constraints, error control mechanisms - and error correction mechanisms in particular - show a trade-off between added complexity and computation and increased communication. In our approach we apply energy consumption constraints to the error control mechanisms in order to *enhance energy efficiency under various wireless channel conditions*. In a wireless environment these conditions not only vary dynamically because the physical conditions of a wireless system can vary rapidly, they can also vary because the user moves from an indoor office environment to a crowded city town. Not only the characteristics could have changed, it is even possible that a complete different infrastructure will be used [Smit 98]. The communication interface of the mobile must not only be able to adapt to these situations and provide the basic functionally, it

must also do it energy efficient in all these situations. An adaptive approach to error correcting mechanisms will shown to be energy efficient.

We define the *energy efficiency e* as the amount of data processed divided by the energy that is consumed to process that data[1]. For example in an error correction mechanism the amount of data equals the number of source packets, and the energy consumed is the amount that was needed to calculate the redundant packets.

### *1.1. Error control in communication systems*

In any communication system, there have always been errors and the need to deal with them. Basically there are two methods of dealing with errors:

- *Retransmission*

  Retransmission methods, such as ARQ (Automatic Repeat reQuest) and variants such as Go-Back-N (GBN) or Selective Repeat (SR) recover errors by re-transmitting the erroneously received packet [Lin 84]. The missing packets are retransmitted upon timeouts, or from explicit requests from the receiver.

- *Error correction*

  Error corrective coding can be used to preserve the integrity of received data. With this technique redundant information is sent with the message, such that the receiver can allow some errors in the received message and so eliminates the need for retransmission of the data. This addition of redundancy is called Forward Error Correction (FEC).

Computer communication generally implements a reliable data transfer using either methods or a combination of them at different levels in the protocol stack. Error correcting codes like FEC are mainly used at the data link layer to reduce the impact of errors in the wireless connection. In most cases, these codes provide less than perfect protection and some amount of residual errors pass through. The upper level protocol layers employ various block error detection and retransmission schemes (see e.g. [Shacham 90, Huitema 96]

### *1.2. Scope and related work*

In our study we focus on the energy efficiency of error control schemes for communication protocols on the wireless channel. In particular we concentrate on error control schemes for systems where Quality of Service (QoS) provisioning is a major concern [Borriss 95], e.g. in wireless ATM systems [Prycker 94]. In these systems QoS is used to control and manage the available resources for multimedia traffic. These communications will include video, audio, images, and bulk data transfer. In multimedia traffic important parameters are jitter, delay, reliability, and throughput [Ferrari 92].

Compression methods such as MPEG or MPEG 2, are very sensitive to cell losses, since these errors would cause multiple errors in the structure of the decompressed data, e.g. instead of a few pixels, a series of complete frames could be lost. Error correcting codes can be beneficial in this field to allow more aggressive encoding, and thus save communication resources[2].

In the presence of stringent QoS and energy consumption constraints it is very hard to use retransmission techniques as basic error control mechanism for several reasons.

- First of all, some of the retransmission techniques affect the QoS dramatically when the error conditions become bad. This is mainly caused by the fact that errors on the wireless link are propagated in the protocol stack. A typical example is TCP/IP that misinterprets errors on the wireless link as collisions and responds to losses by invoking congestion control and avoidance algorithms. The result not only unnecessary reduces the link's bandwidth utilization, but also introduces delays and jitter. Furthermore it increases energy consumption due to several reasons (more communication overhead, more transitions, longer communication time, etc.). Several solutions have been proposed [Balakrishnan 96, Mathis 96] such as: end-to-end protocols, where the sender is aware of the wireless link; link-layer protocols, that provide local reliability and

[1] The energy efficiency is the inverse of the costs to process a certain amount of data.

[2] Note that also in this field there exists a trade-off between a reduction in communication versus the added computation.

shields the sender from wireless losses; and split-connection protocols, that break the end-to-end connection into two parts at the base station. However, the focus with these solutions is mainly on increasing the throughput, and not on preserving QoS and energy efficiency.

- Secondly, classic ARQ protocols overcome errors by re-transmitting the erroneously received packet, regardless of the state of the channel. Although in this way these retransmission schemes *maximize the performance* – as soon as the channel is good again, packets are received with minimal delay - the consequence is that they expend energy. In probing mechanisms, as proposed by [Zorzi 97], the transmitter enters a probing state when the channel conditions deteriorate. In this state the transmitter sends a short probing packet every so often, so that is able to detect when the channel conditions improve. Although this provides an improvement in the energy efficiency, it also introduces a slightly greater delay and jitter. When little delay is allowed, then ARQ performs worse than error correction mechanisms [Zorzi 98]. On the other hand, when the tolerable delay is large enough, ARQ outperforms error correction mechanisms, since the residual error probability tends to zero in ARQ with a much better energy efficiency than error correction methods. In ARQ retransmissions occur only when needed, and bandwidth is not wasted a priori as in FEC. Solutions to provide a predictable delay at the medium access control layer by reserving bandwidth for retransmission are available [Figueira 98], but might spoil bandwidth.
- Another often neglected side effect in ARQ schemes is that the round-trip-delay of a request-acknowledge can also cause the receiver to be waiting for the acknowledge with the receiver turned 'on', and thus spoiling energy. Only with a properly designed energy efficient MAC protocol this effect is negligible. However, nowadays the most MAC protocols only consider throughput and performance as important design criteria, and energy efficiency is no item (the main energy constraints come from the maximal allowable current for a PC-Card [Awater 97]).
- Areas of applications that can benefit in particular from error correction mechanisms are *multicast applications* [Nonnenmacher 96]. Some of these applications, e.g. audio and video-conferencing tools, tolerate losses with a relatively graceful degradation of performance. Others, such as electronic whiteboards or diffusion of information over the network, like electronic newspapers, require reliable delivery of all data [Rizzo 97]. Even if the QoS requirement is not that demanding, insuring the QoS for all receiving applications is very difficult with retransmission techniques since multiple receivers can experience losses on different packets. Individual repairs are not only extremely expensive, they also do not scale well to the number of receivers. Reducing the amount of feedback by the use of error correction, leads to a simple, scalable and energy efficient protocol.
- Finally, another area in which FEC can be beneficial is for the *control messages* of a wireless link, e.g. the control messages for a Medium Access Protocol (MAC). Although these control messages might be tolerable for errors, the throughput can be reduced dramatically in case of errors. For example, when in a TDMA scheme - where traffic control information is being transmitted from base-station to mobile - the control message is lost, then no data-transfer during one time frame can take place. When the traffic on the link has QoS guarantees, then the occurrence of one single bit error in the control message might have dramatic effects.
  Several simple replications for ALOHA have been considered that minimize the expected delay of a successful transmission and improve the throughput-mean-delay characteristics [Birk 98]. In MAC protocols where contention is a significant issue (like ALOHA), adaptive retransmission techniques which try to avoid transmission during bad channel periods, yield a good energy efficiency [Chockalingam 98]. These techniques however can easily misinterpret single bit errors for collision, slow down the link throughput and thus reduce the QoS.

These are some considerations, we have no intention to provide a complete list. A more detailed comparison of the performance ARQ and FEC techniques has been done by many researchers (e.g. [Zorzi 98]), and will not be part of our research. The main conclusion that can be drawn from these considerations is that although good designed retransmission schemes can be optimal with respect to energy efficiency, they can introduce intolerable low performance in delay, jitter and bandwidth to fulfill the required QoS of the application. Although error correction schemes in most cases consume more energy due to increased computation and communication, they can provide the constant quality and stringent delay provisions required for multimedia traffic.

### *1.3. Overview*

The goal of this research is to determine which error-correcting mechanisms should be used in a wireless environment with stringent QoS, and which parameters they should apply to obtain the most energy efficient solution for a variety of error conditions in a wireless environment. In particular we will study the simple parity code, the well-known Reed-Solomon code, and the less known EVENODD code [Blaum 95] that was designed for a system of redundant disks (RAID). We concentrate on an implementation in software. An additional goal is to determine which parameters should be used when the cost for communication is incorporated as well.

First we will provide in section 2 the general strategy that we will use for error correction. We start with the error model that we use, and will argue that the network communication layers do not see the raw physical channel, but a modified channel. The residual channel characteristic after the physical and link layer processing is based on *erasures of packets.* The errors that occur are mostly burst of errors of up to 50 ms. The error correction mechanism therefor has to reconstruct erased packets in a stream. Section 3 gives a short overview of error correcting codes. After a short introduction of the selected codes, a relation of the energy efficiency is determined. Section 4 shows the results of an implementation of the three mechanisms on a general-purpose processor. The energy efficiency factor of our implementation and the pre-conditions for it will be determined for all three mechanisms. The section finishes with a comparison (4.6). The results from this section are used in section 5 to determine the energy efficiency of a minimal system consisting of a general-purpose processor and a wireless interface. As an example we determine the energy efficiency of error control on such a system with a WaveLAN card. We will finish with some conclusions in section 6.

## 2. General strategy

### *2.1. The error model*

Wireless networks have a much higher error rate than the normal wired networks. The errors that occur on the physical channel are caused by phenomena such as signal fading, transmission interference, user mobility and multi-path. The most relevant errors that occur are related to block errors that have a bursty nature. Burst errors are sequences of corrupted or lost bits covering a period of up to a few hundred milliseconds, which degrade the quality of the low bandwidth wireless channels.

In this study we do not deal with the physical layer or on the mechanisms involved like coding and power control. Our starting point is on the interface between the physical transceiver and the higher-level communication system. Most wireless transceiver hardware of wireless systems has some error correction mechanisms build-in to overcome or reduce the impact of the errors on the physical channel. However, it is usually inefficient to provide a very high degree of error correction. Thus in many cases such mechanisms provide less than perfect correction, and some residual errors pass through. The network communication layer therefore does not see the raw physical channel, but a channel modified by the physical layer that has a residual error characteristic [Zorzi 96].

The basis for most currently designed wireless systems will likely be packet switching communication schemes that manage data transfer in blocks that contain multiple symbols (or bits). In an ATM system the basic data block size is one ATM cell, or 53 bytes. Depending on the network interface and the medium access (MAC) protocols being used, the packet size may be greater, and involve ATM adaptation layer (AAL) frames of multiple ATM cells. Lower level ATM layers (network layer or adaptation layer) generally provide error detection by using checksums (e.g. Cyclic Redundancy Checksums (CRCs). Corrupted packets are identified and discarded. For the transmission of unidirectional video data through an ATM network, the CCITT (ITU-T) has specified an AAL1 transport scheme which uses forward error correction (FEC). Retransmission of ATM cells is hard to implement and needs substantial buffering, since the ATM system requires an in-order reception of cells.

The residual channel characteristic after the physical and link layer processing is therefore based on *erasures*, i.e. missing packets in a stream [Rizzo 97]. Erasures are easier to deal with than errors, since the exact location of the missing data is known. In the following we will discuss error-correcting mechanisms in general, and consider erasure-correcting codes as a special case.

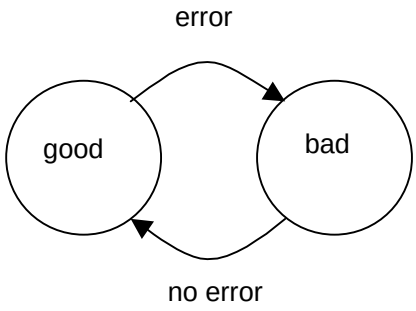


**Figure 1: The two-state error model.**

We use a two-state error model, as shown in Figure 1, to represent the quality of a wireless channel between the mobile and the base-station. The two states in the model represent error and error-free periods. During the *good* state bits are transmitted without errors. The *bad* state occurs when bits are lost or corrupted. Such a model is well suited to capture the occurrence of burst errors. The quality of the channel (*good* or *bad*) in such a model depends only on the current time. This differs from a model that non-stochastically corrupts a small percentage of the bits in transit over the wireless channel.

In our experiments we use this model by using a distribution for the periods spent in the *good* and *bad* states. We analyze and perform experiments with error correcting mechanisms that can recover from a wide range of error rates, and can tolerate a large error-burst. The bad state period of a realistic system in a wireless office building environment can be up to 50 ms [Zorzi 98]. On a 2 Mb/s wireless system one ATM cell takes about 0.2 ms, so the bad period time in such a system is equivalent to bursts of up to 250 ATM cells.

### *2.2. The encoding packet model*

The basis for our model will be packet switching communication schemes that manage data transfer in blocks that contain multiple symbols (or bits). Errors that might have occurred on the physical layer result either in erasures or in packet errors. Both errors and erasures are assumed to be detected by some detection technique, and the whole packet will be discarded. Error detection is often based on a checksum using a Cyclic Redundancy Code (CRC). We will ignore the overhead involved with error detection in our model, since the computation of the CRC requires little overhead, and is already provided in the protocol or hardware of many communication systems.

Figure 2 shows a graphical representation of the error correction mechanism. The sender collects a number of *source data packets* in a buffer. When the buffer is full, the data is encoded, and the encoded data is transmitted. The receiver is able to reconstruct the original data from a subset of the encoded data, and so can allow the erasure of some packets.

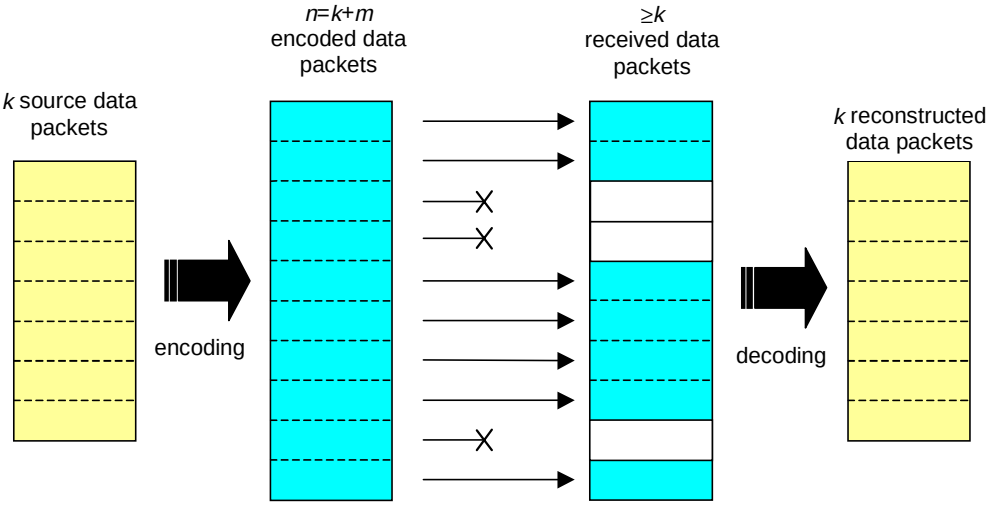


**Figure 2: Graphical representation of error correction.**

If the encoded data includes a verbatim copy of the source data, the code is called a *systematic code*. The advantage of a systematic code is that it allows a simple reconstruction of the source data when

there are few losses. In this research we will consider systematic codes that consist of the original source data packets supplemented with redundant packets. We will denote the number of source packets as *k*, the packet size as *S*, the number of redundant packets *m*, and the number of encoded packets as *n*. Such a code is called an (*n*,*k*) code and allows the receiver to recover from *m (=n-k)* losses in a group of *n* encoded packets. This structure can be seen as an (*S*) x (*k* + *m*) array in which the columns represent a packet of length *S*, the first *k* columns represent the source data packets, and the last *m* columns represent the redundant packets. All packets together build up one *frame*. Figure 3 gives a graphical representation of this scheme.

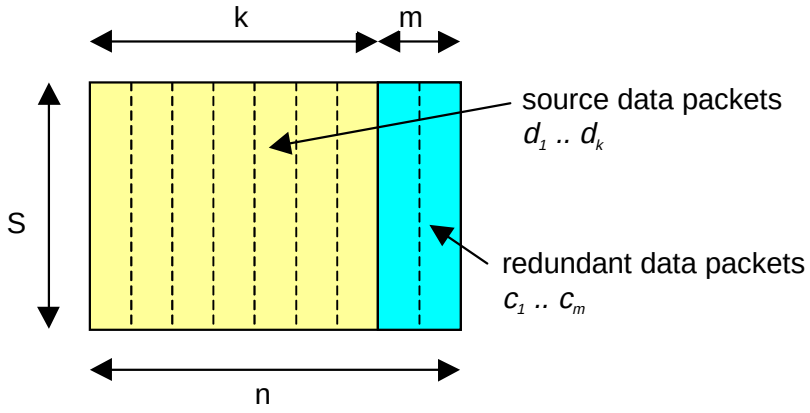


**Figure 3: Representation of an encoded frame.**

The calculation of the contents of redundant packet $c_i$ requires the function $F_i$ applied to all source data packets *d*. The coding method breaks up each packet into symbols. The size of a symbol can be anything, but generally it is a multiple of 8 bits. The coding functions $F_i$ operate on a symbol-by-symbol basis. However, to make the notation simpler, we can assume the coding function to operate with a packet as a computational element. Thus we have to compute *m* redundant packets $c_1, \ldots, c_m$ from *k* data packets $d_1, \ldots, d_k$, in such a way that the erasure of *m* packets can be tolerated. To compute redundant packet $c_i$ we apply function $F_i$ to all data packets:

$$c_i = F_1(d_1, d_2, \ldots, d_k) \qquad (1)$$

We define the encoded packets $e_1, .., e_n$ to consist of a concatenation of the source packets and the redundant packets.

Suppose that up to *m* packets are erased, then we can reconstruct the original data by applying some function $F^{-1}$ on at least *k* correctly received data packets *e* to recover an erased packet. The not correctly or erased packets are not used.

$$d_i = F_i^{-1}(e_1, e_2, .., e_n) \quad \text{(on k correct packets from } e_1 .. e_n) \qquad (2)$$

The problem of error correction can be defined with our packet model as defining function *F* to calculate the redundant packets; and finding function $F^{-1}$ to reconstruct the lost packets when up to *m* packets have been erased.

## 3. Error correction techniques

Error-correcting techniques are generally based on the use of error detection and correcting codes. These codes have been studied for some time now. They were first discovered abstractly in 1945 when Claude Shannon proved a theorem which stated that, even with a noisy channel, there exist ways to encode messages such that they have an arbitrarily good chance of being transmitted safely. The first construction of error-correcting codes was in 1947, at Bell Labs, by a mathematician named Richard Hamming. He devised ways to encode the input to a computer such that the computer could correct isolated errors. Soon after, Marcel Golay generalized Hamming's construction, and constructed codes that corrected one error per word which used *p* symbols (rather than just 0 and 1) for *p* prime. He also constructed two codes that correct multiple errors.

Nowadays error-correcting techniques are being used in many fields of information processing, and particularly in telecom systems. Further practical applications include storage in computer memories (RAM), and magnetic and optical storage systems. Error-correcting codes have also been used in

'Redundant Arrays of Inexpensive Disks' (*RAID*) systems where batteries of small inexpensive disks were used to combine high capacity, bandwidth, and reliability all at low cost [Patterson 88]. Since then the technique has been used to design multi-computer and network file systems with high reliability and bandwidth [Hartman 93].

The RAID failure model has some similarities with our error model that is based on erasures. In both models $k$ blocks of source data are encoded to produce $n$ blocks of encoded data, in such a way that any subset of $k$ encoded blocks suffices to reconstruct the source data. Such a code is called an ($n$,$k$) code. So, an ($n$,$k$) code can tolerate the erasure of ($n$-$k$) blocks of data, and the maximum error rate is thus $(n - k) / n$.

For small number of devices $n$, one redundant device is often sufficient for fault-tolerance. This is the RAID level 5 configuration, and the coding technique is called '$n$+1-parity'. With $n$+1-parity, the $i$-th byte of the redundant device is calculated to be the exclusive-OR (XOR) of the $i$-th bit of each data device. If any one of the devices fails, it can be reconstructed with the XOR of the remaining $n$ devices.

As $n$ grows, the ability to tolerate multiple failures becomes important. The most general technique for tolerating $m$ simultaneous failures with $m$ redundant devices is a technique based on *Reed-Solomon* coding [MacWilliams 77, Berlekamp 68]]. This technique however requires computation over finite fields and results in a complex implementation. An attractive compromise might be a scheme like EVENODD that only requires simple exclusive-OR operations and that it is able to tolerate two failures. In this paper we will compare the energy efficiency of these mechanisms.

The size of the block is for a RAID system generally much larger than for communication systems. In RAID systems the block size can be a disk sector, but may even be a complete disk; whereas communication systems deal with packet sized data. The block size that is used in telecommunication systems is even smaller: from several bytes to maximal a few hundred bytes.

The major difference is that in a RAID system a major concern is the *update-penalty*, which is the cost required to update the encoded data when data is written to a device. In communication the only two processes are encoding and decoding, and the major concerns are complexity and memory requirements.

In the following section we will give an overview of three error correcting codes and the complexity of the algorithms. In particular we will look at parity coding ($k+1$,$k$), EVENODD coding ($k+2$,$k$), and Reed-Solomon coding ($n$,$k$).

### *3.1. Parity coding*

Parity coding is the simplest form of error correcting codes. It is an ($k+1$,$k$) code so that it allows one failure that can be reconstructed.

*Data encoding*

We can describe ($k+1$,$k$) code with one redundant packet $c_1$ as follows: to compute $c_1$, we take the exclusive-OR (also called the parity) of the data symbols:

$$c_1 = F_1(d_1, d_2, \ldots, d_k) = d_1 \oplus d_2 \oplus \ldots \oplus d_k \qquad (3)$$

*Data recovery*

If packet $d_j$ fails, then each word may be restored as the parity of the corresponding words on the remaining packets:

$$d_j = F_i^{-1}(e_1, e_2, .., e_k) = d_1 \oplus \ldots \oplus d_{j-1} \oplus d_{j+1} \oplus \ldots \oplus c_1 \qquad (4)$$

*Energy efficiency of parity coding*

Both encoding and decoding requires $k$ exclusive-OR operations to be performed on each data symbol with size $s$. Thus the overhead to process $k$ source packets is $O(k.s)$. Therefore, the energy efficiency $e_p$ (which can be seen as the inverse of the overhead costs) is:

$$e_p = E_p / k \qquad (5)$$

($E_p$ is the energy efficiency factor for parity coding in a particular implementation). This shows that the most energy efficient coding is the smallest parity code that involves two data source packets and one redundant packet. Although this gives also a good error correction rate, it also involves a significant communication overhead.

### *3.2. EVENODD coding*

In this section we will give a short overview of the EVENODD algorithm based on the paper by Blaum et al. [Blaum 95], and focus on the impact of a possible implementation.

The EVENODD coding scheme is an (*k+2,k*) code. It was intentionally meant for tolerating two failures in RAID architectures, but we will show that it is also suitable in communication systems.

The basic scheme requires the number of source packets *k* to be a prime number. However, this is not a very hard constraining requirement. If we want to use a non-prime number for *k*, then we can take the next prime following the required *k*, and assume the extra imaginary packets to contain zeros. The packet size *S* is for simplicity restricted to contain (*k* – 1) symbols. This restriction is not hard too because a symbol can be of any size, and we can introduce imaginary data also to fill the columns to the required size. The imaginary zeros introduce some overhead when the packet size is small and *k* is large. As we assume the basic communication element to be an ATM cell, a symbol will likely be a multiple of 53 bytes.

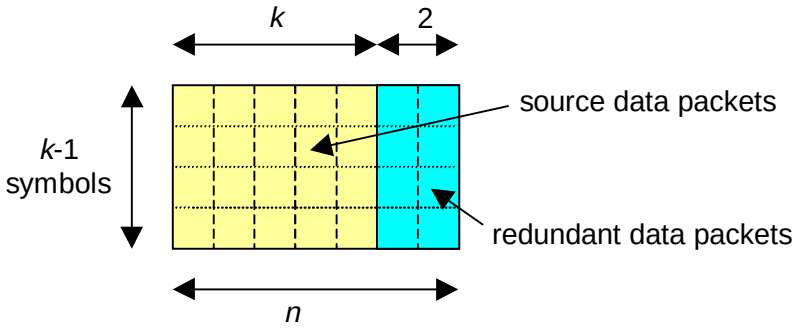


**Figure 4: Basic EVENODD frame.**

So, the packet model can be considered as an (*k* – 1) x (*k* + 2) array, *k* a prime number, such that the symbol $d_{ij}$, $0 \le i \le (k-1)$, $0 \le j \le (k-2)$, is the *i*-th symbol in the *j*-th packet. The last two packets (*k* and *k* + 1) are the redundant packets. In the description of the decoding we will assume an imaginary 0-row after the last row, i.e. $d_{k-1,j} = 0$, $0 \le j \le k-1$.

In the following description we will use the following notation: $j = |n|_m$ as $j = n \pmod m$ with $0 \le j \le m-1$.

*Data encoding*

There are two types of redundancy: *horizontal* redundancy and *diagonal* redundancy. The redundant value of each is stored in a redundant packet. The value of the horizontal redundancy (stored in packet *k*) is the exclusive-OR of packets 0, 1, …, *k* –1. This is thus exactly the same as with simple parity encoding. Packet (*k*+1) carries a diagonal redundancy. This is calculated using the exclusive-OR of the diagonals of the matrix and *P*. *P* is calculated via the exclusive-OR of a special diagonal. So, for example the first redundant symbol in redundant packet *k* +1, denoted as $d_{0,k+1}$, is calculated with: $d_{0,k+1} = P \oplus d_{0,0} \oplus d_{k-2,2} \oplus \ldots \oplus d_{1,k-1}$. Since the source packet matrix is an (*k* – 1) x (*k*) matrix, one diagonal is not calculated. This diagonal (formed by the indices (*k*-2,1), (*k*-3,2), …, (0,*k*-1)) is used to determine the value of *P*. *P* determines whether the diagonal redundancy is either even or odd.

| | *0* | *1* | *2* | *3* | *k-1* | *k* | *k+1* |
|---|---|---|---|---|---|---|---|
| *0* | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| *1* | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| *2* | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| *k-2* | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

**Figure 5: EVENODD coding example for k=5.**

An example of an encoded frame with symbols of one bit is shown in Figure 5. Notice that $P = d_{3,1} \oplus d_{2,2} \oplus d_{1,3} \oplus d_{0,4}$. , and e.g. $d_{2,6}$ is obtained as follows: $d_{2,6} = P \oplus d_{2,0} \oplus d_{1,1} \oplus d_{0,2} \oplus d_{3,4}$.

*Data recovery*

Data encoded with the EVENODD scheme is able to recover maximal two packet erasures. Note that recovering is also possible for finer grained erasures: i.e. not all erasures need to be in the same two packets. Depending on the topology of the symbol erasures up to $2(k-1)$ *symbol erasures* can be restored. When we evaluate and compare the performance of the coding schemes, we will discuss this property in more detail (see sections 3.4 and 4.6).

Reconstruction when only one packet is erased (and assuming it is not a redundant packet) is simple as the missing packet can be retrieved using the exclusive-OR of the packets.

When two packets $i$ and $j$ , $0 \leq i < j \leq k+1$, are erased, then we have four cases:

- $i = k$ *and* $j = k + 1$

  Both redundant packets are erased, but since they are redundant no recovery is needed.

- $i < k$ *and* $j = k$

  Which means that a data packet and the horizontal redundant packet are lost. To recover packet $i$ we first determine $P$ by taking the exclusive-OR of symbol $| i - 1 |_k$ of the redundant packet $(k + 1)$ and the symbols of the diagonal belonging to symbol $(k, i)$. (Remember that we assume an imaginary 0-row after the last row, i.e. $a_{k-1,j} = 0$, $0 \leq j \leq k-1$) Then the missing symbols can be recovered using the exclusive-OR of $P$, symbol $| i - 1 |_k$ of the redundant packet $(k + 1)$, and the non-erased symbols of the diagonal corresponding with the missing symbol. So, when for example $i$=2, then $P = d_{1,k+1} \oplus ( d_{0,1} \oplus d_{k,2} \oplus \ldots \oplus d_{2,k-1})$. Symbol (0,2) can be found using: $d_{0,2} = P \oplus d_{1,k+1} \oplus (d_{2,0} \oplus d_{1,1} \oplus d_{k,3} \oplus \ldots \oplus d_{3,k})$.

- $i < k$ *and* $j = k + 1$

  One data packet and the diagonal redundant packet are lost. We can recover packet $i$ by taking the exclusive-OR of the valid data-packets.

- $i < k$ *and* $j < k$

  Both erased packets are data packets. In this case we cannot reconstruct the data using the parities separately, as in the previous three cases.

  First we need to compute the diagonal parity $P$ by calculating the exclusive-OR of the symbols in packets $m$ and $m + 1$. Then we calculate the horizontal and vertical syndromes $S0$ and $S1$. A syndrome consists of $m-1$ symbols. The syndromes are basically calculated using the exclusive-OR of the horizontal packets, and the diagonal packets. Now the missing packets can be reconstructed. First the scheme recovers a symbol using the exclusive-OR of one element of the diagonal syndrome $S1$ with a correct symbol. Then the just recovered symbol is used to recover an other symbol by calculating its exclusive-OR with $S0$. Recovery can formally be written as: $d_{s,j} \leftarrow S1_{|j+s|m} \oplus d_{|s+(j-i)|m,i}$ and $d_{s,i} \leftarrow S0_s \oplus d_{s,j}$ in which $s \leftarrow |s-(j-i)-1|_m$ and $s$ initially equals to –1.

*Energy efficiency of EVENODD coding*

*Encoding* for the horizontal parity packet requires $k$ exclusive-OR operations to be performed on each data item. The diagonal parity requires $k + 1$ exclusive-OR operations, including the calculation of S. This makes the total encoding complexity $O( s. (2k +1))$. Therefore, the energy efficiency $e_{eoe}$ (that relates the amount of data per energy consumed) is:

$$e_{eoe} = E_{eoe} / (2k + 1) \qquad (6)$$

($E_{eoe}$ is the energy efficiency factor for encoding EVENODD).

The *decoding overhead* is dependent on the number of erased packets, which packets are erased (i.e. whether redundant packets are involved or not), the number of source packets *k*, and on the size *s* of a packet. We will only deal with the complexity of the erasure of two data packets as this is the most common case, and the most complex.

This case has three main steps. First, calculating the diagonal parity *P* requires $O( 2 . s )$ exclusive-OR operations. Then the syndromes *S0* and *S1* are calculated, requiring $O( s . (k-1) )$ and $O( s . k )$ XOR operations. Finally the reconstruction takes another $O( 2 . s )$ XOR operations. This makes the total decoding complexity $O( 2k + 3 ) s )$, which are basically all XOR operations. When $E_{eod}$ is the energy efficiency factor for decoding EVENODD then the energy efficiency equals:

$$e_{eod} = E_{eod} / (2k + 3) \qquad (7)$$

### *3.3. Reed-Solomon coding*

In this section we will give a short overview of the Reed-Solomon algorithm based on the paper by Plank [Plank 97], and focus on the complexity and energy efficiency of a possible implementation. More details and an excellent tutorial on Reed-Solomon coding can be found in the paper.

The Reed-Solomon coding scheme is an (*n,k*) code. There are three main aspects of the Reed-Solomon algorithm: the use of the Vandermonde matrix to calculate the redundant packets, the use of Gaussian elimination to recover from failures, and the use of Galois fields to perform arithmetic.

*Data encoding*

We defined $c_i = F_1(d_1, d_2, \ldots, d_k)$, or in other words, if we represent the data and redundant packets as the vectors *D* and *C*, and the functions $F_i$ as rows of the matrix *F*, then the state of the system adheres to: $F\,D = C$.

A simple and effective matrix *F* is a Vandermonde matrix in which $f_{i,j} = j^{\,i-1}$. The calculation of the redundant packet then requires some simple matrix arithmetic:.

$$\begin{vmatrix} f_{1,1} & f_{1,1} & . & . & f_{1,k} \\ f_{2,1} & f_{2,2} & . & . & f_{2,k} \\ . & . & & & . \\ f_{m,1} & f_{m,2} & . & . & f_{m,k} \end{vmatrix} \begin{vmatrix} d_1 \\ d_2 \\ . \\ d_k \end{vmatrix} =$$

$$\begin{vmatrix} 1 & 1 & 1 & . & 1 \\ 1 & 2 & 3 & . & k \\ . & . & . & & . \\ 1 & 2^{\,m-1} & 3^{\,m-1} & . & n^{\,m-1} \end{vmatrix} \begin{vmatrix} d_1 \\ d_2 \\ . \\ d_k \end{vmatrix} = \begin{vmatrix} c_1 \\ c_2 \\ . \\ c_m \end{vmatrix}$$

*Data recovery*

We can reconstruct the original data by applying some function $F^{-1}$ on at least *k* correctly received data packets *e*. First we define the vector *E* to contain the original data packets *D* and the redundant packets *C*. Matrix *A* is defined to consist of the *identity matrix I* and the generator matrix *F*.

Then we have a system that adheres to the equation ($A\,D = E$). When a packet is erased, we reflect this

$$A = \begin{vmatrix} I \\ F \end{vmatrix} \qquad E = \begin{vmatrix} D \\ C \end{vmatrix}$$

by deleting the corresponding row from the vector $E$ and the matrix $A$, resulting in a new matrix $A'$ and a new vector $E'$ that adheres to the equation A' D = E'. Recovery of the original data is possible by solving the linear system: $A' D = E' \rightarrow D = (A')^{-1} E'$ using Gaussian elimination.

*Arithmetic with Galois fields*

A major concern is that the domain and range of our computations are binary words of a fixed length $w$. Since practical algebra implementation does not use infinite precision real numbers, we must perform addition and multiplication over a *finite field* of more than $k + n$ elements. A field is closed under addition and multiplication, which means that the results are still field elements. This allows us to make exact computations on field elements without the requirement of expensive - in both computation time and bits – computer arithmetic. Fields with $q = p^w$ elements, with $p$ prime and $w > 1$ are called extension fields or Galois Fields denoted as $GF(p^w)$.

Operations on extension fields are simple in the case $p = 2$. The elements of $GF(2^w)$ are integers from zero to $2^w - 1$. Addition and subtraction of $GF(2^w)$ are simple exclusive-OR operations. Multiplication and division are more complex and require two mapping tables, each of length $2^w$. These two tables map an integer to its logarithm and its inverse logarithm in the Galois field. If the number of field elements is not too large a table for the multiplication can be used as well. Note that a multiplication in $GF(2^8)$ already requires a 64 kB lookup table!

*Energy efficiency of the Reed-Solomon algorithm*

The *encoding overhead* depends on the number of source packets $k$, on the number redundant packets $m$ (= $n – k$) and on the size $S$ of a packet. The encoder requires $k$ source data packets to produce each encoded packet, and thus the encoding overhead to process $k$ source packets is $O((n-k).k.S)$. Therefore, an approximation of the energy efficiency of encoding $e_{rse}$ equals:

$$e_{rse} = E_{rse} / (n - k) \,.\, k \qquad (8)$$

in which $E_{rse}$ is the energy efficiency factor for encoding with Reed-Solomon.

The *decoding overhead* is more complicated as it involves two parts: the Gaussian elemination, and the reconstruction. This requires a matrix inversion to be performed once, and then a matrix multiplication for each reconstructed packet which is maximal $m$. The matrix multiplication requires $O(k)$ operations for each reconstructed data item, or a total of $O((n-k).k.S)$ operations per block of $k$ packets. Although the matrix inversion requires $O(k.(n-k)^2)$ operations per $k$ packets, the cost of matrix inversion becomes negligible for reasonable sized packets. In the experiments this will be shown clearly.

So, if we assume the number of reconstructed packet to be equal to ($n$-$k$) then an approximation of the energy efficiency of decoding $e_{rsd}$ equals:

$$e_{rsd} = E_{rsd} / (n - k) \,.\, k. \qquad (9)$$

in which $E_{rsd}$ is the energy efficiency factor for decoding with Reed-Solomon.

### *3.4. Error-rate and burst-size*

The data-model of a frame consists for all mechanisms of an $S$ x $k$ array of data packets, supplemented with $n$-$k$ redundant packets, each packet has size $S$. When all these parameters can be easily changed, like in our software implementation, it becomes interesting to know what to choose for these parameters. The two main parameters in our error model are the error-rate and the size of the error-burst.

The maximal *error-rate* that a mechanism can correct is dependent on the number of source packets $k$ and on the number of redundant packets $n$-$k$. A mechanism with an ($n$,$k$) code can tolerate the erasure of ($n$-$k$) blocks of data. The correctable error-rate is thus $(n - k) / n$. The correctable error rate that is used by the error correcting code is normally determined by the bad state probability $e$ of a channel.

The *maximal burst size* $B_{max}$ is the size of the burst that can be corrected by the error correction mechanism. The value depends on the allowable error rate, the packet size $S$ and the number of redundant packets. An ($n$,$k$) code can tolerate ($n$-$k$) erasures, and thus the maximal burst size of an ($n$,$k$) code is $(n-k-1)$ packets. This only holds for ($n$-$k$) >1, since when only one redundant packed is used, then no guarantees can be given because a small burst of only two bytes can cause two packets

to be corrupted and erased. In general it is of no use to have a larger packet size $S$ than $B_{max}$. Only when $S$ would become too small, then the implementation overhead would impose a larger $S$. A smaller $S$ gives a lower latency and needs less buffer-space.

There exist a relationship between error rate and burst size. For example, what is the difference between a (64,32) code with a maximum error rate of 32/64 = 0.5, and a (4,2) code that also has a maximum error rate of 0.5? Both codes can handle a bad state probability $e$ of 0.5. However, the maximal burst size of the (4,2) code is one packet, and the maximal burst size of the (64,32) code is 31 packets. If the packet size used is the same, then the maximal burst size of the (64,32) code is bigger. However, if the packet size can be adapted, then the maximal burst size of both codes can be made equal. The difference that remains is *flexibility*. The (64,32) code is able to tolerate many small bursts (basically up to 32), whereas the (4,2) code can basically tolerate only one burst. (Note that several smaller bursts that occur in the same packet result in the erasure of only one packet.) Another difference is that the code with larger $k$ allows a much more precise adaptation to the current error situation. It is however questionable whether such precise numbers are available in a real system.

The EVENODD mechanism uses a ($k$-*1*) x ($k$+*2*) data array of symbols. This structure offers the EVENODD mechanism a somewhat similar flexibility in burst size without the need of a large number of redundant packets $m$. It is however more restricted. The ($k$+*2*,$k$) code allows up to two erased packets to be recovered using the data in the remaining non-erased packets. This scheme further also allows for a more finer-grained error reconstruction: depending on the 'topology' of the errors, several *symbol* erasures – not necessarily in the same two packets - can be recovered. With the mechanism maximal *two symbol erasures on a row* can be restored. For example, with an EVENODD (4,2) code, not only two erased packets can be restored, but also up to two symbol erasures on a row can be restored, with a maximum of 2.($k$-1) = 6 symbol erasures. This however is still much less than the Reed-Solomon coding, but the difference becomes smaller for larger $k$. When applied to a communication protocol, then the symbol-size can be made equal to the communication packet, e.g. an ATM packet.

# 4. Implementation and results

## *4.1. Reconfigurable computing*

Implementation of the three described error correction mechanisms can be done in several ways. All algorithms can be implemented either in software or in hardware. Most current implementations of Reed-Solomon coding are based on custom chips and provide a high throughput (e.g. LSI logic L64711 RS codec supports up to 50 Mbits/s [LSI logic]). Recently a FPGA manufacturer (Xilinx) already provides support for implementing a Reed-Solomon codec in an FPGA [RS-Xilinx]. As much of current communication design already is performed using DSPs, error correction codecs are also implemented on DSPs.

Reconfigurable computing is targeted for applications that are application specific, but would benefit from a higher degree of flexibility in the design than ASICs can offer, either in performance or in energy consumption. Reconfigurable computing lies between the two extremes of a full custom ASIC (fixed function, low operation diversity) and a general-purpose processor (diverse function and operation).

Error correcting is an area that is quite well suited for adaptation.

- First of all, there are many error correcting algorithms possible, each offering a considerable amount of variability in the parameters. Using adaptation and reconfiguration it is possible to balance on the trade-off between performance and cost.
- Furthermore, in real applications, system designers will determine the error correction capability by investigating the worst case of communication channel reliability in any environment. However, the channel only remains in the worst case for a small fraction of total time, and thus spoils energy used for the error correction. These systems and applications could benefit from reconfiguring to perhaps a different algorithm, a different code length, or a different error correcting capability.

For different application or user requirements we will need different parameters. Adaptive error correction has shown to be much better than a single code scheme in terms of bandwidth utilized and

in terms of a profit function which combines the bandwidth utilized and the deadline miss rate [Elaoud 98]. In a reconfigurable implementation the system can just download the required design into the machine. In full custom implementation, the designer either has to implement different algorithms and parameters for different requirements in one or multiple chips, or has to use a more complicated architecture for adaptive code-word length. The custom chip can not have programmability of error correction capability, because the architecture will have to be changed for different parameters. The disadvantages of re-configurable components such as FPGAs are their cost in high quantities, their typical higher consumption of power than custom ICs, and the limited capabilities.

### *4.2. Software implementation*

In the next sections we will show the results of a software implementation with a general-purpose processor of the three error correction mechanisms. Such an implementation is the most flexible solution and can adapt its algorithm and its parameters very quickly to changing environments. We are aware of the fact that a software implementation is not the most energy efficient solution, and might not provide enough performance. However, there exist many applications and systems that do not need high performance and cannot use the capabilities and advantages of real reconfigurable computing, simply because its lack the required hardware. For example, a notebook computer can also benefit from an energy efficient solution, although not the most optimal possible when the required hardware would be available.

A software implementation has also a number of specific advantages in favorite compared to a hardware solution:

- The use of a microprocessor allows *very rapid adaptations* to varying error conditions (burst size, frequency) and required QoS from applications. The adaptation to perform can be applying another error control scheme, or adapting some parameters of the error control scheme.
- A software implementation allows us to experiment with a large set of error control schemes, and experience in ‘real life’ how applications behave. When we have a good feeling of the behavior of the schemes, then we could compose a subset of error control schemes that is suitable to be implemented in hardware, either in an FPGA, a DSP, or a custom chip.
- The error control can easily and efficiently be embedded in various layers of the communication protocol where the data is buffered anyway. Little extra overhead is expected with a good engineered and well-integrated error correction mechanism.
- A standard processor also allows the use of relatively large memories, and thus allows for much larger block lengths than standard custom chips (that typically allow a block length of up to 255 bytes). In a wireless office environment burst errors of 1 to 100 ms can be expected. To handle these large erasures at relatively high speed (say 2 Mb/s), a large block size is needed. We provide results of error correcting schemes that can tolerate bad periods of 1-256 ATM cells. To tolerate longer bad periods the system requires more memory, more computation, and also introduces more latency.

Due to the lack of good measurement possibilities with a general-purpose computer, we will assume the energy consumed by the algorithm to be linearly related to the amount of time needed for the processor to do its calculations. We neglect the properties of some processors to shut down parts of the processor when not in use. However, when comparing the algorithms for use on one type of machine, the results are useful.

Reducing the power consumption of the processor saves more than just the energy of the processor itself. When the processor is doing less work, there is less activity for other components of the computer, such as memory and the system bus. For example, when the processor on the Macintosh DUO 270c is off, not only is the 1.15 W of the processor saved, but also an additional 1,23 W from other components [Lorch 95]. However, in our study the additional energy consumed by the external memories and its interface will be neglected because they are very system dependent. Furthermore, in current implementations of memory architectures it has no or little influence whether the memories are in use or not (i.e. contain useful data or not). We concentrate on the general strategy, and will provide general guidelines. We will give two examples, using the Pentium Pro 133 processor and the StrongARM SA-1100, 133 MHz.

All three implementations are written in *C* and are portable across many platforms. The measurements were performed on a Toshiba 220CS notebook that has a Pentium Pro 133 processor and runs Windows 95. The results are only to be used as a reference, since the actual performance depends on items like memory speed, cache size, quality of the compiler, operating system, etc. The code is written straightforward, and only uses the most obvious optimizations (like the use of a multiply lookup table for Reed-Solomon coding). Handcrafted code that makes good use of the specific features of the processor (like registers, caching and the use of special instructions) might achieve significant speedups.

In the following sections we will present the results of our measurements for each mechanism. For decoding we will always assume that the maximal allowable number of erasures have occurred, which is the worst-case. We will start with a presentation of the encoding and decoding times in μseconds per kByte. Then we will validate our analysis of the coding mechanisms by determining the influence of the amount of data that is processed and the influence of the overhead involved for small number of source packets *k*. These results will give us the constraints under which we can determine a qualification the energy efficiency for the coding mechanisms.

### *4.3. Parity coding implementation*

In this section we present the results of the measurements performed with our implementation of the parity coding. In our measurements we have used a data model of a frame that consists of a *k* x *k* matrix of symbols. The symbols can be of any size, but normally are a multiple of a byte. Each column represents a packet. The parity is calculated using the symbols in one row of the frame. In our error model we assume packet erasures only. When a packet is erased, it can be recovered using the data in the remaining non-erased packets. Note that this scheme also allows for a more finer-grained error reconstruction: any single symbol that is erased in a row can be recovered. A frame model that consists of a 1 x *k* array of packets is also possible, but a *k x k* array resembles the EVENODD scheme most.

This (*k+1,k*) coding scheme allows us only to change the number of source packets *k*. However, every implementation of an algorithm has some overhead involved. The effect of this overhead is getting less when the packet size over which the code has to work is enlarged, because it will be amortized over more data. The effect of using larger packet sizes is shown in Figure 6. It shows the symbol size *s* (in bytes) versus the normalized encoding time $T_e$ (in μs/kB) for *k*=3 and *k*=16. The results for decoding are almost the same, and hence not shown in the figure.

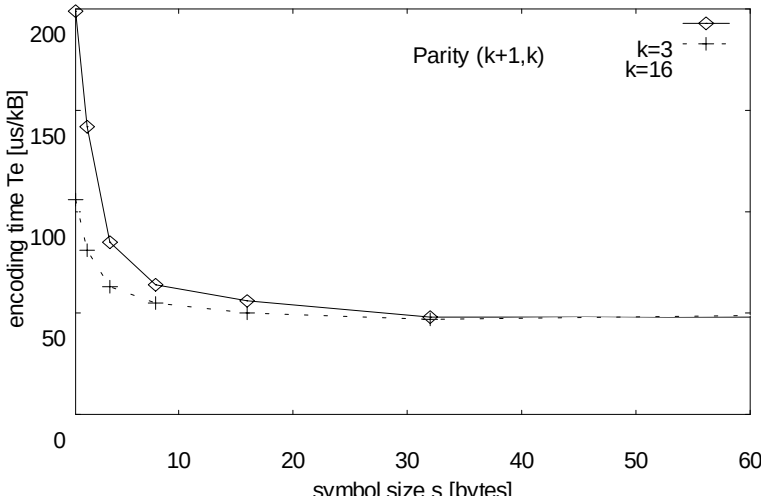


**Figure 6: Encoding time $T_e$ vs. symbol size *s*.**

The figure also shows that the number of source packets *k* hardly influences the encoding time, especially for larger symbol size.

The overhead to process *k* source packets is theoretically linearly dependent on *k*, so

$$T_e = k \,.\, C_e \qquad (10)$$

and

$$T_d = k \,.\, C_d \qquad (11)$$

The results of the measurements, and also shown in Figure 7 for $k$=8, have provided the values of $C_e$ and $C_d$ for various symbol sizes. From the figure can be determined that the encoding time $T_e$ is linearly dependent on $k$ provided that symbol size $s$ is large enough (more than about 8 bytes). Decoding time $T_d$ has almost the same characteristic.

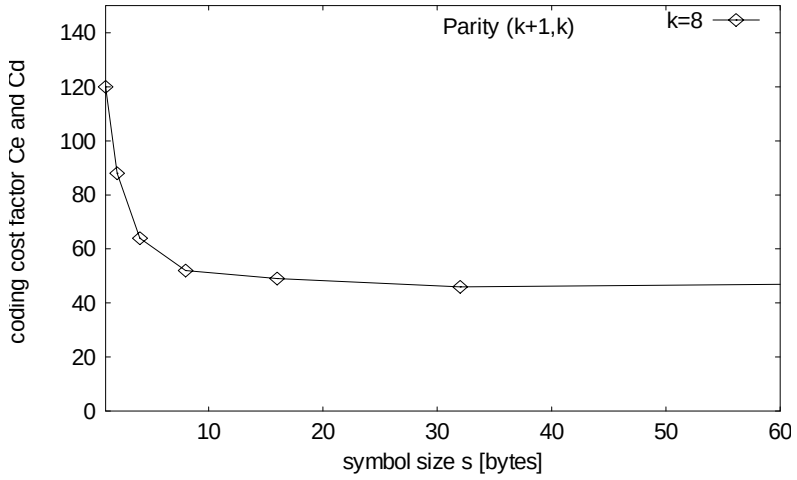


**Figure 7: Encoding factor $C_e$ vs. symbol size $s$.**

As shown in section 3.1 the energy efficiency for both encoding and decoding is $e_p = E_p \,/\, k$. To be able to have some quantitative values for the energy efficiency of error correcting mechanisms, we need to know the energy efficiency factor $E_e$ for various values of symbol size $s$, and determine the influence of the implementation overhead. Figure 8 shows the values of $E_e$ versus $k$, for various values of $s$.

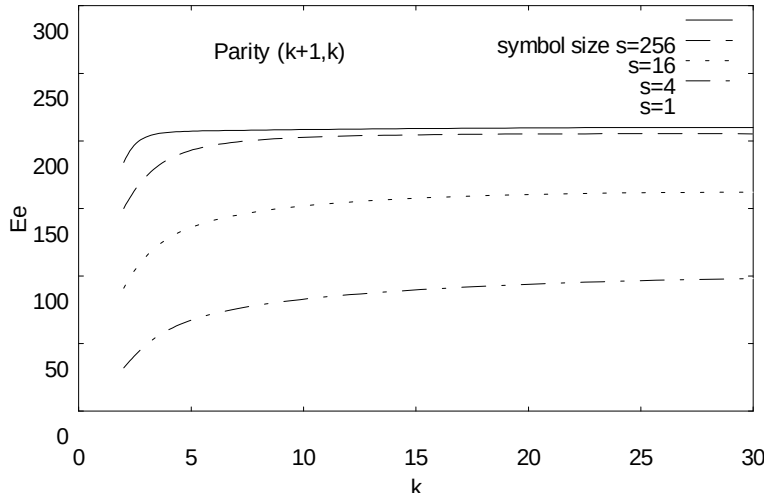


**Figure 8: Energy efficiency factor $E_e$ vs. the number of source packets $k$ for various symbol sizes.**

The figure shows that the energy efficiency factor $E_e$ is almost stable for $k$>3 , and approaches its maximum (205) with a symbol size $s$ of 16 bytes.

Measurements on decoding show similar results. It is stable for $k$>3 and is a bit more efficient with its maximum of 230 reached with a symbol size of 16 bytes.

So, parity encoding provides a mechanism that for small symbol sizes and a small number of source packets already has stable characteristics. We can safely use the following values when comparing the energy efficiency factor of the mechanisms: $E_e$ = 205, and $E_d$ = 230. Note again that the Parity scheme only allows the reconstruction of one erased packet.

### *4.4. EVENODD coding implementation*

The data model that we have used in our *C*-code implementation of the EVENODD coding resembles the basic model described in section 3.2 added with the imaginary 0-row. So, we use a (*k*) x *(k* + 2) array, *k* a prime number, of symbols with size *s*. Each column represents a packet. The last two columns (*k* and *k* + 1) are the redundant columns. The symbols can be of any size, but normally are a multiple of a byte.

In our error model we assume packet erasures only. Since it is a (*k+2,k*) code up to two erased packets can be recovered using the data in the remaining non-erased packets. Note that this scheme, just like the Parity scheme, allows for a more finer-grained error reconstruction: any single symbol that is erased in a row can be recovered.

In the measurements we will vary the number of source packets *k* and the symbol size *s*. In Figure 9 is shown that the effect of the implementation overhead is getting less when the packet size over which the code has to work is enlarged, because it will be amortized over more data. A better performance is also reached due to the effect of caching since the same data is used several times. The figure shows the symbol size *s* (in bytes) versus the normalized encoding time $T_e$ (in µs/kB) for *k*=5 and *k*=13. The graphs show that the effect of the overhead is already small with a symbol size of 16 bytes. It also shows that the number of source packets *k* hardly influences the encoding time. The measurements also show, although not shown in the graph, that the encoding time has reached its minimum for *k*=13. A larger *k* hardly decreases the encoding time.

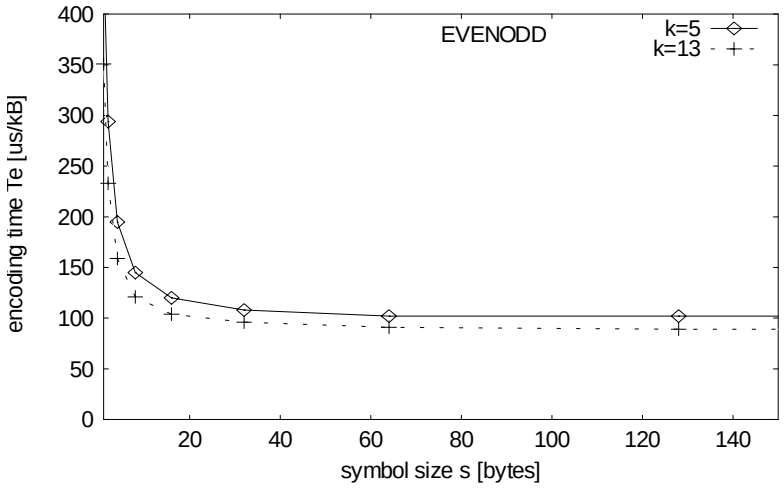


**Figure 9: Encoding time $T_e$ vs. symbol size *s*.**

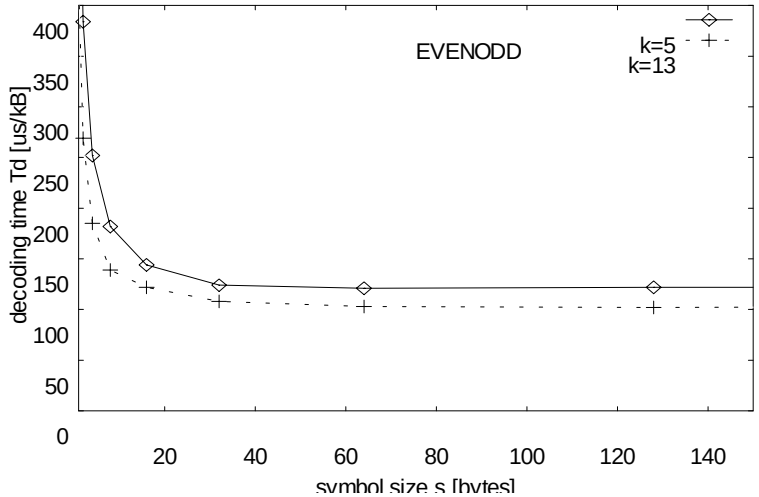


**Figure 10: Decoding time $T_d$ vs. symbol size *s*.**

Decoding takes more time (see Figure 10), and the effect of the implementation overhead is small for symbols size of 32 bytes. The decoding time reaches is maximum for $k$=13. The encoding time $T_e$ and decoding time $T_d$ can be formulated using (6) and (7) as

$$T_e = C_e . (2k + 1) \qquad (12)$$

and

$$T_d = C_d . (2k + 3) \qquad (13)$$

In Figure 11 is shown (for $k$ equals to 13) how $C_e$ and $C_d$ are influenced by the implementation overhead caused by the size of a symbol. When the symbol size is large enough (more or equal to 8 bytes), then the values of $C_e$ and $C_d$ remain almost constant. This shows the validity of our analysis of the EVENODD coding, and that our approximations of the energy efficiency are correct.

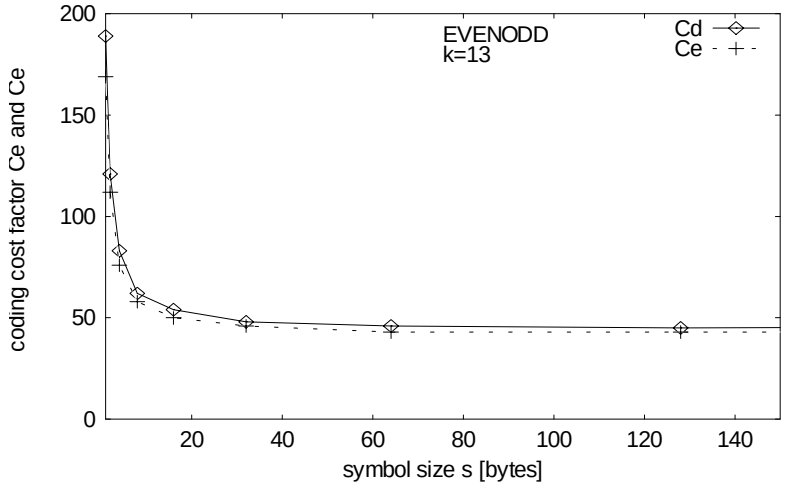


**Figure 11: Coding cost factor $C_e$ and $C_d$ vs. symbol size $s$.**

From (6) and (7) we know that the encoding energy efficiency $e_{eoe}$ and the decoding energy efficiency $e_{eod}$ is:

$$e_{eoe} = E_{eoe} / (2k + 1)$$
$$e_{eod} = E_{eod} / (2k + 3)$$

To quantify $E_{eoe}$ and $E_{eod}$ Figure 12 and Figure 13 show the characteristics of $E_{eoe}$ and $E_{eod}$ versus $k$, for various values of $s$.

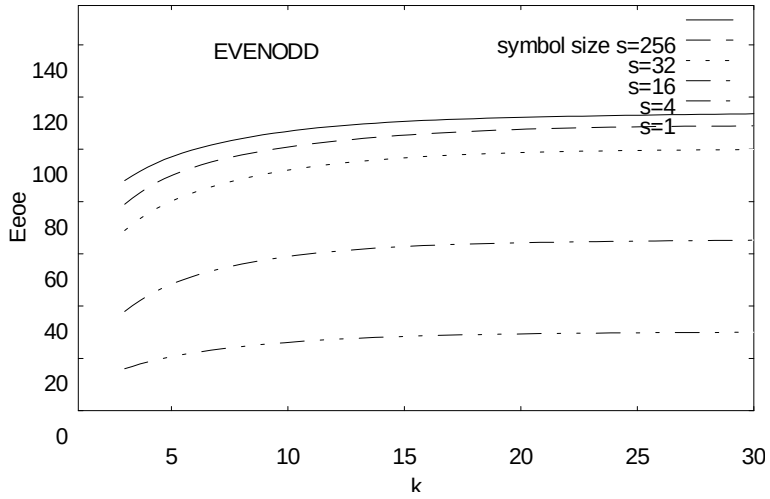


**Figure 12: Energy efficiency factor $E_{eoe}$ vs. the number of source packets $k$ for various symbol sizes (in bytes).**

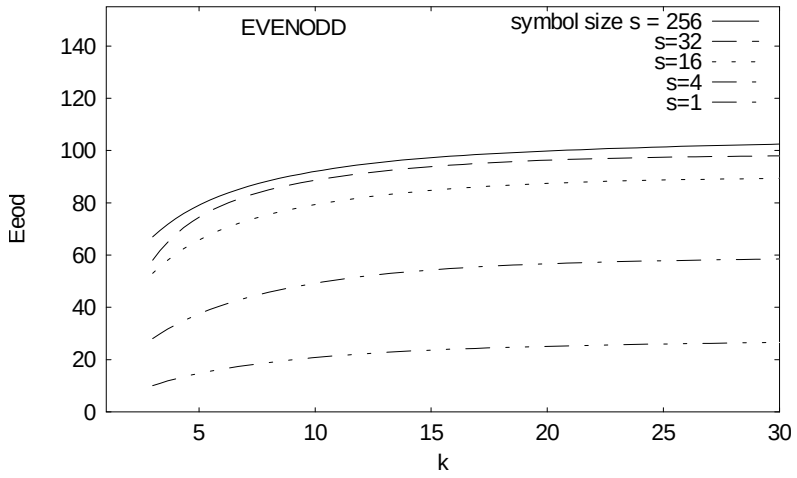


**Figure 13: Energy efficiency factor $E_{eod}$ vs. the number of source packets *k* for various symbol sizes (in bytes).**

The figure shows that the energy efficiency factor $E_{eoe}$ is almost stable for $k>5$ , and approaches its maximum for a symbol size *s* of 32 bytes.

So, EVENODD encoding provides a mechanism that for reasonable small symbol sizes (*s* = 32 bytes) and a small number of source packets (*k* = 5) already has stable characteristics. Under these conditions we can approximate the following values when comparing the energy efficiency factor of the mechanisms: $E_{eoe}$ = 110, and $E_{eod}$ = 100. The energy efficiency can become a little better for higher values of *k* and larger symbol sizes.

### *4.5. Reed-Solomon coding implementation*

In this section we present the results of the measurements performed with our implementation of the Reed-Solomon coding. The code is based on an implementation from Rizzo, Karn and others [Rizzo 98]. The data model that we have used in our measurements is an (*S*) x *(k + (n-k)*) array of symbols with size *s*. Each column represents a packet with size *S*. The last *n-k* columns are the redundant columns. The code supports $GF(2^w)$, for any *w* in the range of 2..16. In the measurements we have used *w* = 8 since this gives the maximum efficiency because most operations can be executed using lookup tables. So, the symbols size *s* in our measurements will be one byte. We have also used a lookup table for the multiply operations.

In our error model we assume packet erasures only. Since it is a (*n*,*k*) code, up to *n-k* erased packets can be recovered using the data in the remaining non-erased packets. Note that this scheme does *not* allow for a more finer-grained error reconstruction like EVENODD or Parity. Only whole erased packets can be restored.

Since the symbol size *s* in our measurements equals to one byte, we can only vary the packet size *S*. We will choose *S* to be multiples of ATM cells sizes (53 bytes) since this will be our main target. Note that although this is a much larger quantity than single bytes, it does not influence our main goal - the determination of the energy efficiency of the coding mechanism - since we only need to know when the implementation overhead has become negligible. So, in the measurements we will vary the packet size *S*, the number of redundant packets *n-k*, and the number of source packets *k*.

First the encoding and decoding times will be evaluated. Figure 14 shows the encoding time $T_e$ (in µs/kB) versus the packet size *S* (in number of ATM cells) for a number of different values for *n-k* and *k* = 64. It shows clearly that the encoding time is practically independent on the packet size. A packet size of already one ATM cell involves not many overheads. Decoding is more influenced by the packet size as shown in Figure 15. This is mainly caused by the cost of matrix inversion which cost $O(k \cdot l^2)$, where *l* is the number of packets which must be recovered, which in our measurements we assume to be equal to *n-k*. The influence is small only for packet sizes greater or equal to 8 ATM cells.

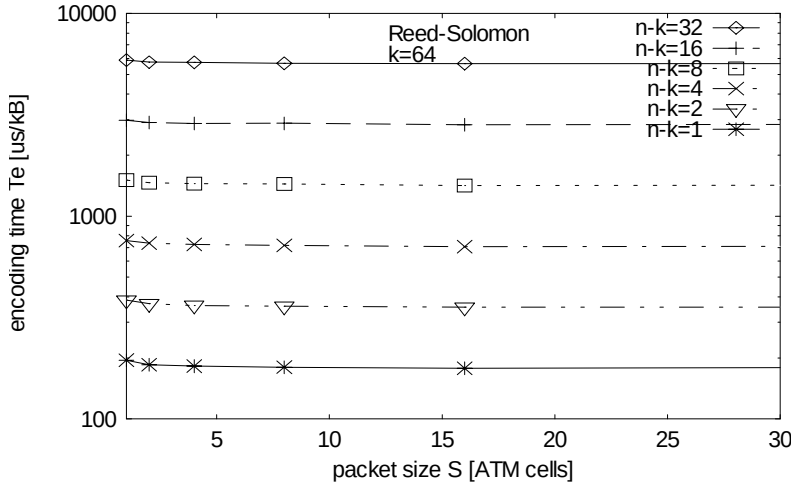


**Figure 14: Encoding time $T_e$ vs. symbol size $s$ for various numbers of redundant packets.**

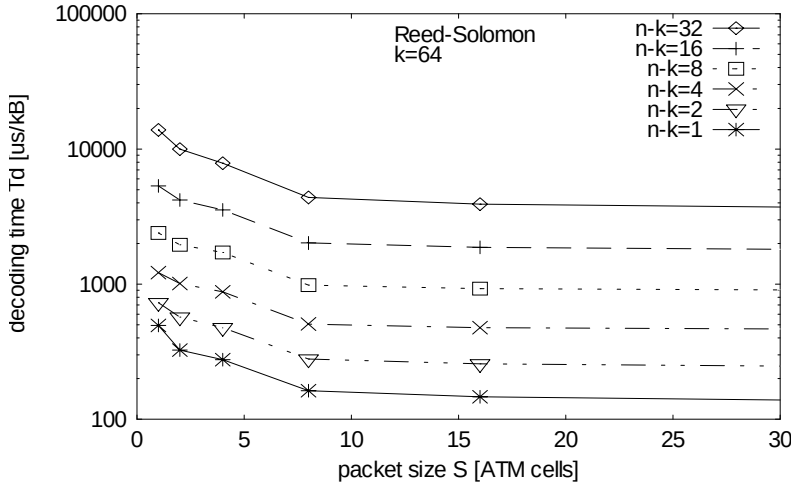


**Figure 15: Decoding time $T_d$ vs. symbol size s for various numbers of redundant packets.**

Both figures also show that the influence of the number of redundant packets is large. It was argued in section 3.3 that the cost of calculating a redundant packet is constant for reasonable sized packets and larger $k$. The encoding time $T_e$ and decoding time $T_d$ can be formulated using (8) and (9) as

$$T_e = C_e \,.\, (n\text{-}k) \,.\, k \quad \text{and} \quad T_d = C_d \,.\, (n\text{-}k) \,.\, k$$

In Figure 16 is shown (for $k$ equals to 32, and $n$-$k$=2) how $C_e$ and $C_d$ are influenced by the implementation overhead due to the packet size $S$. The value of $C_e$ remains constant when the packet size is greater or equal to two ATM cells. Decoding cost factor $C_d$ becomes constant after a packet size of 8 ATM cells. This shows the validity of our analysis of the Reed-Solomon coding.

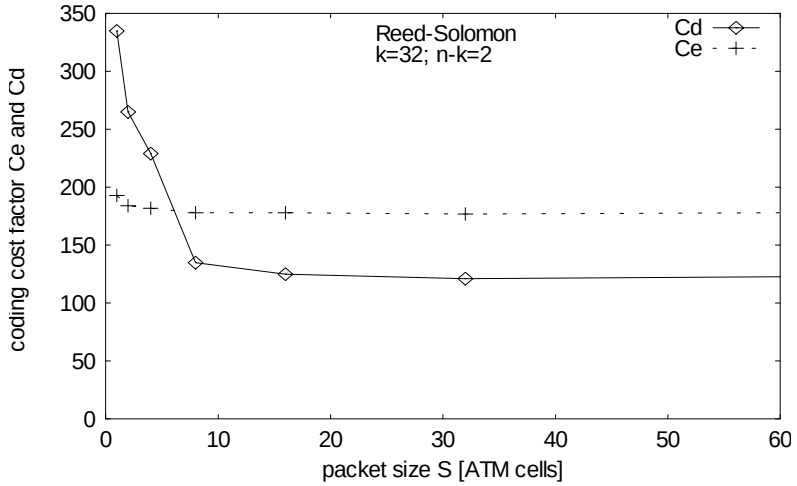


**Figure 16: Coding cost-factor $C_e$ and $C_d$ vs. packet size $S$.**

From (8) and (9) we know that the encoding energy efficiency $e_{rse}$ and the decoding energy efficiency $e_{rsd}$ equals:

$$e_{rse} = E_{rse} / (n - k) \,.\, k$$

$$e_{rsd} = E_{rsd} / (n - k) \,.\, k$$

We will now present a qualification of the energy efficiency factors $E_{rse}$ and $E_{rsd}$. Figure 17 shows these characteristics versus $k$, for various values of $S$.

As discussed above, the energy efficiency of encoding is hardly influenced by the packet size. Therefor only one graph is shown in the figure for encoding. The figure shows further that the number of source packets also hardly influences the efficiency. Encoding is already stable for small values of $k$ and for all packet sizes.

The decoding graphs shows that the energy efficiency becomes reasonable stable for small values of $k$. The efficiency is much more influenced by the packet size $S$: the energy efficiency of decoding reaches it maximum for packet size greater or equal to 16 ATM cells.

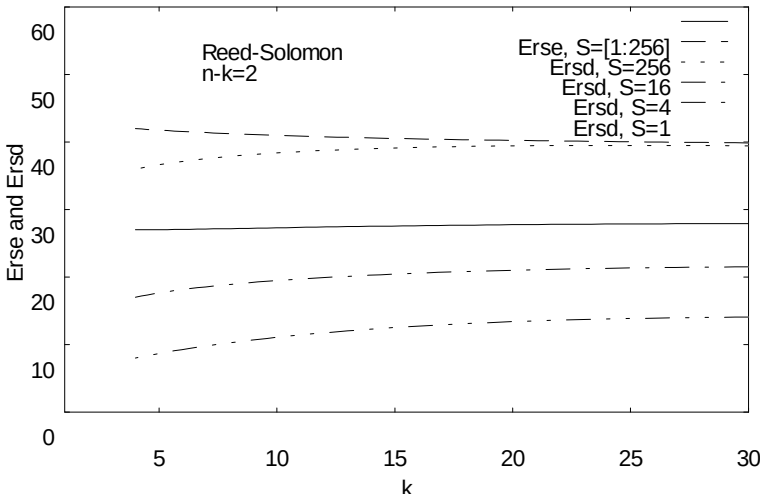


**Figure 17: Energy efficiency factor vs. number of source packets for various packet sizes (in ATM cells).**

So, Reed-Solomon encoding provides a mechanism that for packet sizes starting from 16 ATM cells and a small number of source packets ($k$ = 5) has stable characteristics. Under these conditions we can use the following values with $n$-$k$=2: $E_{rse}$ = 28, and $E_{rsd}$ = 40.

### *4.6. Comparison*

We can compare the EVENODD and the Reed-Solomon mechanisms for $n$-$k$=2, and use a pseudo ($k$+2,$k$) mechanism for Parity (see section 5.3). The energy efficiency factors for the Parity mechanism will be divided by two. All mechanisms reach a constant performance with constant overhead at small values of $k$ and for small data sizes. To summarize we have the following results:

| *mechanism* | $E_{encoding}$ | $E_{decoding}$ | *Minimal k* | *Minimal data size* |
|---|---|---|---|---|
| Parity | 102 | 115 | 3 | symbol: 16 bytes |
| EVENODD | 100 | 90 | 5 | symbol: 32 bytes |
| Reed-Solomon | 28 | 40 | 5 | packet: 16 ATM cells |

**Table 1: Energy efficiency factors of error correcting codes for $n$-$k$=2.**

The efficiency of the mechanisms will in general be a bit better when larger $k$ and/or symbol sizes are used. Please note again that the numbers must be interpreted with some caution, since they can be very much influenced by the actual implementation and system on which it runs. The results show that the Parity and EVENODD mechanisms perform almost equally well for encoding. The decoding scheme for EVENODD is more complicated, and less efficient. Note however that this is completely due to the fact that the Parity mechanism is not a real ($k$+2,$k$) code.

Reed-Solomon encoding is about four times as inefficient as the others are. Decoding more than two times as inefficient.

These results are in line with the comparison in [Blaum 95] that compared the number of exclusive-OR operations between encoding with the Reed-Solomon scheme and the EVENODD scheme. They however also incorporated the number of exclusive-OR operations needed to perform the multiply that is needed for Reed-Solomon, whereas we have implemented this with a lookup table (see section 4.5). They showed that the complexity of implementing EVENODD in a disk array with 15 disks is about 50% of the one required when using the Reed-Solomon scheme. The difference in complexity grows when the number of disks increases, e.g. with 43 disks the Reed-Solomon scheme requires nearly 5 times as many operations.

Comparing the minimal size of a data item (either a symbol or a packet) is more complicated and depends on the choice of the packet size for EVENODD and Parity. When the packet size is chosen to be one column (just like our Reed-Solomon implementation), then the minimum size of a packet for EVENODD equals 32 ( $k$ - 1 ). The minimum packet size for EVENODD is thus smaller than for Reed-Solomon for approximately $k$<26. E.g. when $k$=7, then the minimal packet size for EVENODD equals just more than 4 ATM cells, which is much less than the minimum of 16 cells for Reed-Solomon coding.

*Conclusion*

A general conclusion about which mechanism offers the best characteristics in both error-correction as energy efficiency is hard to give, since it depends on many factors.

The three mechanisms have different characteristics and capabilities. The Reed-Solomon code is attractive because it is the most general technique capable of tolerating $n$-$k$ simultaneous failures. Reed-Solomon can be used quite well when a high fault tolerance is required with a maximum flexibility in burst size. It is further attractive since it has a lot of flexibility in its parameters. A disadvantage is that it is more complex than the other schemes and involves operation over finite fields which makes it less energy efficient than other techniques like EVENODD and Parity.

The EVENODD mechanism consists of simple exclusive-OR operations only. When $n$-$k$=2 then EVENODD is much more efficient than Reed-Solomon coding. Especially when the bad-state probability is not high and with reasonable sized $k$ (say > 3) then EVENODD might offer an energy efficient alternative, because it also provides a flexibility in burst-size and error-rate, comparable with Reed-Solomon. The EVENODD mechanism allows the reconstruction of erased *symbols*, which increases its flexibility. The Reed-Solomon mechanism used is only capable of handling erasures of packets. Since the size of a packet is generally much larger that the size of a symbol as used in EVENODD, this gives a further reduction in energy consumption for decoding when the burst-error

size is small. For example, consider a situation with a bad-state probability *e* of less than 0.15 and with an error-burst-size of 1-10 ATM cells. An EVENODD (13,11) code gives an error correcting capability of 2/13=0.15 and can allow maximal two packet erasures and 20 symbol erasures. When the symbol is one ATM cell, then it can tolerate one large error-burst of 10-20 consecutive ATM cells, or maximal 20 single ATM erasures. A comparable Reed-Solomon code that can allow a burst of 10 erasures is at least a (65,55) code with a packet size of 1 ATM cell. A disadvantage of the EVENODD code, although not very hard, is the fact that *k* must be a prime. If we want to use a non-prime number for *k*, then we can assume extra imaginary packets that contain zeros.

The results of Parity mechanism show, although it has a high efficiency, that it is only little more efficient than EVENODD. Since it is only an ($k$+1,$k$) code, it is not an appropriate choice.

# 5. Error correction energy efficiency of a system

## *5.1. Introduction*

In the previous sections we have investigated the *computational* energy efficiency of three error correction mechanisms. We will now consider the *energy efficiency of a system* in which also the energy consumption of the communication interface is incorporated. We will only consider the data that is actually transmitted, and not incorporate additional costs involved with the wireless interface like turning 'on' and 'off' the transceiver, sending extra control data, etc. These matters are dealt with in e.g. the medium access control layer. A more precise analysis would require these costs to be incorporated. However, these costs are very dependent on the underlying protocols and operating system, and the energy savings capabilities of the system. Furthermore, note that the additional costs needed for FEC are expected not to be very high, the additional cost with an ARQ mechanism might be much higher. An ARQ mechanism needs a potentially long time in which the receiver must be turned 'on' waiting for an acknowledge. To have a clean comparison we will thus in our analysis only use the energy needed for the actual data transfer.

Error correction mechanisms for wireless communication involve computational overhead and communication overhead at both the transmitter and the receiver side. This is overhead in time, but also overhead in energy consumption. In our context we mainly focus on the *energy overhead U*.

The transmitter overhead $U_t$ is composed out of two elements, the encoding overhead $U_e$, and the transmitter communication overhead $U_{tx}$

$$U_t = U_e + U_{tx} \qquad (14)$$

At the receiver side the overhead $U_r$ is composed out of two elements, the decoding overhead $U_d$, and the receiver communication overhead $U_{rx}$

$$U_r = U_d + U_{rx} \qquad (15)$$

The overhead of the encoding and decoding depends on the error correction method, the number of source packets *k*, the number of redundant packets *m*, and on the packets size *S*. This overhead is discussed in the previous sections.

## *5.2. Communication overhead*

The *communication overhead* mainly depends on the number of additional bits that are transmitted. The number of redundant bytes equals the number of redundant packets *m* multiplied by the packet size *S*, and thus the total communication overhead of *k* source packets is $O(mS)$. So the transmitter communication overhead per source packet of an ($n$,$k$) code (with $m = n-k$) equals:

$$U_{tx} = C_{tx} \cdot S \cdot (n - k) / k \qquad (16)$$

We have defined the *energy efficiency e* as the amount of data processed divided by the energy that is consumed to process that data. In an error correction mechanism the amount of data equals the number of source packets, and the energy consumed is the amount that was needed to calculate the redundant packets. The extra communication is caused by the extra redundant packets that needs to be send.

So, the communication energy efficiency of transmitting an *(n,k)* redundant code $e_{tx}$ (with $n-k>0$) equals:

$$e_{tx} = E_{tx} \cdot k / (n - k) \qquad (17)$$

in which $E_{tx}$ is a constant representing the energy efficiency factor for transmitting. Similarly, the energy efficiency of the receiving side (assuming that no packets are lost) equals:

$$e_{rx} = E_{rx} \cdot k / (n - k) \qquad (18)$$

The maximum loss rate that can be sustained equals $(n – k) / n$. So, note that although the energy efficiency increases when less redundant packets are transmitted, the maximum loss rate decreases.

### *5.3. Error correction overhead*

In the previous sections we have determined the energy efficiency of three error correction mechanisms. To summarize:

| | |
|---|---|
| Parity ($k+1$,k): | $e_p = E_p / k$ |
| EVENODD encoding ($k+2,k$): | $e_{eoe} = E_{eoe} / (2k + 1)$ |
| EVENODD decoding ($k+2,k$): | $e_{eod} = E_{eod} / (2k + 3)$ |
| RS encoding/decoding ($n,k$): | $e_{rs(e/d)} = E_{rs(e/d)} / (n - k) \cdot k$ |

All mechanisms show better energy efficiency when the number of source packets $k$ and the number of redundant packets $(n-k)$ is small.

The mechanisms have different error correcting capabilities: Parity coding can sustain one packet erasure in a frame, the EVENODD mechanism is capable of recovering from two packet erasures in a frame, and only the Reed-Solomon mechanism is capable of recovering $(n-k)$ packet erasures. To be able to compare the mechanisms for any value of $(n-k)$, we need an approximation of the energy efficiency of Parity coding for $(n-k) > 1$ and EVENODD coding for $(n-k) > 2$. This simplification is only needed to show the general trade-off that exists between communication and error correction. The data packet model uses a frame of $k$ source packets and $(n-k)$ redundant packets. For simplicity we build a frame using parity coding consisting of $(n-k)$ sub-frames. We approximate the error correcting capability of such a frame as having a $(n,k)$ code. Similarly we approximate the error correcting capabilities of EVENODD to a $(n,k)$ code by using $(n-k) / 2$ EVENODD sub-frames. Note that this is a simplification, since the reconstruction capability of erasures in a real $(n,k)$ system does not depend on the location of the erasures in the frame. For example, the error correcting capabilities of Reed-Solomon for $(n-k) > 1$ is better than the pseudo-frame build with parity sub-frames, since it can resolve any $(n-k)$ erasures on the whole block, whereas the pseudo $(n,k)$ frame build from the parity code can sustain only $(n-k)$ erasures when they are not in the same sub-frame.

So, the energy efficiency $e_{ec}$ of a Reed-Solomon code and of the simplified energy efficiency of Parity and EVENODD coding (simplification when $n-k>1$, respectively $n-k>2$) for $k$ large is approximately (with $E_{ec}$ the energy efficiency of error correction):

$$e_{ec} = E_{ec} / (n-k) \cdot k \qquad (19)$$

### *5.4. Trade-off*

Relation (19) shows that when the number of redundant packets for all mechanisms is set to a constant value, the energy efficiency increases when the number of source packets $k$ decreases. The energy efficiency of *communication* $e_{com}$ on the other hand shows a greater efficiency when the ratio between the number of source packets $k$ and the number of redundant packets $(n-k)$ is large (with $E_{com}$ the energy efficiency of communication):

$$e_{com} = E_{com} \cdot k / (n - k) \qquad (20)$$

In Figure 18 both functions are plotted (with $(n-k)$ constant), and the trade-off is shown clearly. The value of $k$ where the communication overhead equals the error correction overhead depends on the implementation of the coding and on the energy efficiency of the communication.

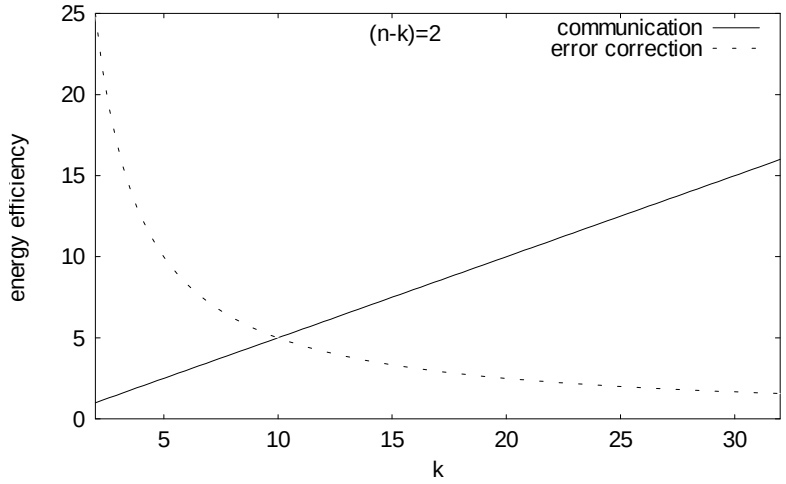


**Figure 18: System energy efficiency *e* vs. number of source packets *k*.**

Note that the maximum loss rate that can be sustained equals $(n - k) / n$. So, when determining the most energy efficient value of $k$ for a constant number of redundant packets ($n$-$k$), $k$ must always be below the required value.

The energy efficiency of a system composed of a transceiver for the communication and a codec for the encoding and decoding of the packets equals to:

$$1/e_{tot} = 1/e_{com} + 1/e_{ec} \qquad (21)$$

or

$$e_{tot} = \{ (n - k) / E_{com} \,.\, k \;+\; (n - k) \,.\, k / E_{ec} \}^{-1} \qquad (22)$$

This implies a relation between $k$ and the energy efficiency factors of error correction and communication. We define the energy efficiency ratio $R$ as the ratio between the energy efficiency factor of error correction divided by the energy efficiency factor of communication.

$$R = E_{ec} / E_{com} \qquad (23)$$

### *5.5. Example system energy efficiency*

As stated before, there exists a trade-off between the added error correction capabilities and the increased energy consumption needed to transfer the extra redundant data. Figure 18 showed the general trade-off.

We know from (22) the relation of the system energy efficiency with $k$, $n$-$k$ and the energy efficiency factors $E_{com}$ and $E_{ec}$. In our measurements we compared the energy efficiency of three mechanisms that all used two redundant packets ($n$-$k$=2). The energy efficiency factor $E_{ec}$ that we have determined from our implementation already incorporates the factor $n$-$k$=2, thus with our measured values the total system energy efficiency must be rewritten as:

$$e_{tot} = \{ 2 / E_{com} \,.\, k \;+\; k / R \,.\, E_{com} \}^{-1} \qquad (24)$$

When we translate the measured energy efficiency factors to the real world, then we can determine the energy efficiency of error-correcting mechanisms when applied in a system with real communication devices. As an example we will determine the energy efficiency of a small system consisting of a WaveLAN PCMCIA card as wireless communication device and a Pentium Pro 133 MHz as general purpose processor (the same processor as used in our experiments).

The energy efficiency $e$ is defined as the amount of data processed per energy consumed. We will compare the energy efficiency factors using a rating that indicates the amount of energy consumed to process one byte, using:

$$E = 1 / ( \text{time to process 1 byte [ } \mu s] \,.\, \text{required power [mW] } ) \qquad (25)$$

The WaveLAN interface consumes approx. 1800 mW when transmitting [WaveLan]. The data transfer-rate is 2Mb/s. One byte takes thus 4 µs to process. The energy efficiency rating of WaveLAN to transfer one byte thus equals to: 1 / (4.1800) = 1 / 7200. The Pentium Pro 133 processor takes 14 W [PentiumPro]. As an example we will now calculate *R* for encoding with parity. The time needed to encode 1 kB of data using the Parity mechanism is 47 µs, so one byte takes 47/1024 µs. The energy efficiency rating of encoding with parity is thus: 1 / ( (47/1024) . 14000 ), which is approximately 1 / 642. The energy efficiency ratio *R* between encoding with parity and communication thus equals 7200 / 642 = 11.2 (or in other words using the energy efficiency factors: $E_{ec} = 11.2 \, . \, E_{comn}$).

The energy efficiency ratios for encoding and decoding of all mechanisms are shown in Table 2 (for sufficiently large *k* and data size, and *n*-*k*=2).

| *mechanism* | *R encoding / Tx* | *R decoding / Rx* |
|---|---|---|
| Parity | 11.2 | 12.6 |
| EVENODD | 11.0 | 9.9 |
| Reed-Solomon | 3.0 | 4.4 |

**Table 2: Energy efficiency ratio *R* with a Pentium Pro 133 and WaveLAN.**

The Pentium Pro processor is not optimized for its energy consumption. If we use a different processor, like the StrongARM, then we can expect quite other ratios. The StrongARM SA-1100 processor [SA-1100] has several features and mechanisms to reduce the amount of energy consumed. The 133 MHz version consumes typically 200 mW, and maximal 300 mW. When we assume that – at least for the basic instructions we use - the performance of the PentiumPro and the StrongARM are equal, then the energy efficiency of the StrongARM is 70 times higher. Figure 19 shows the energy efficiency of error correction on a system using WaveLAN with a PentiumPro processor and a fictive StrongARM implementation.

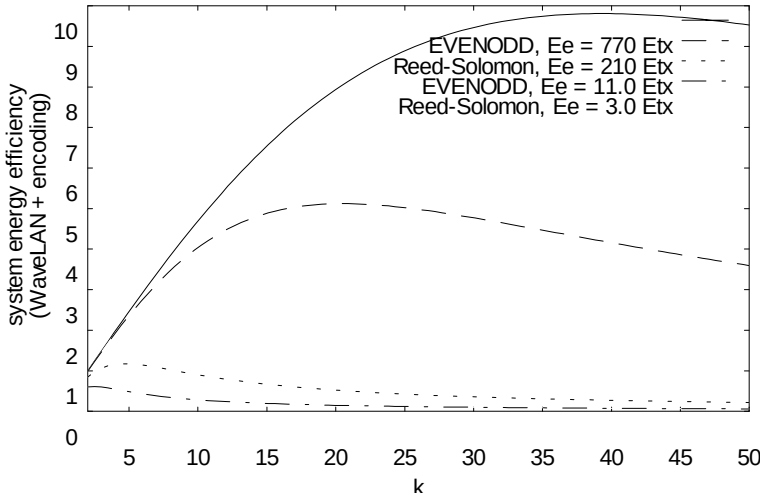


**Figure 19: Energy efficiency of error correction on systems with WaveLAN.**

The figure shows only the transmitting side, thus encoding and sending the data. The graphs with $E_e$=210 $E_{tx}$ and $E_e$ =770 $E_{tx}$ represent the fictive implementation on a StrongARM. The general tendency is that the energy efficiency increases with increasing *k* (because less communication is needed), up to a certain $k_{max}$ where the encoding cost is becoming the limiting factor. When the efficiency of encoding is high (like the StrongARM implementation of EVENODD), then $k_{max}$ is high also. If the efficiency of encoding is low (like the Reed-Solomon encoding on a PentiumPro), then the communication cost is negligible.

This shows that is of no use to have a larger *k* than $k_{max}$, not only because it is less efficient, but also because it has lower error correcting capabilities. On the other side, *k* must be chosen small enough to sustain the bad state probability of the system in a certain environment. This *k* is called $k_{min}$. For *n*-*k*=2 the tolerable error rate is determined by $e<2/(k+2)$, thus $k<2/e-2$. Both $k_{max}$ (for energy efficiency) and

$k_{min}$ (for error correcting capabilities) thus determine the appropriate choice of $k$ in a system. The general consensus that it is profitable to minimize the number of bits over the air-interface [Elaoud 98] is thus not correct and effective when considering the energy efficiency of a system.

### *5.6. General strategy*

It was shown in this section that from energy efficiency perspective it is not always profitable to minimize the number of bits transmitted over the air. There exist an optimum value for $k$ that should be used to have a good fault tolerance at the most energy efficient way.

The value of $k_{max}$ is the maximum of the graph from relation (22). We define the number of redundant packets ($n$-$k$) to be a constant value $m$, and use the same ratio $R$ that defines the ratio in energy efficiency between error correction and communication as in (23), but now assume a normalized $E_{ec}$ valid for one redundant packet. We then can deduce using the first order differential of (22) the following relation:

$$e' = - m / (E_{com} \,.\, k^2) + m / R\, E_{com} = 0 \qquad \Rightarrow$$

$$k = \sqrt{(R)} \qquad (26)$$

Figure 20 shows this relation graphically between the energy efficient value of the number of source packets $k_{max}$ and the ratio $R$.

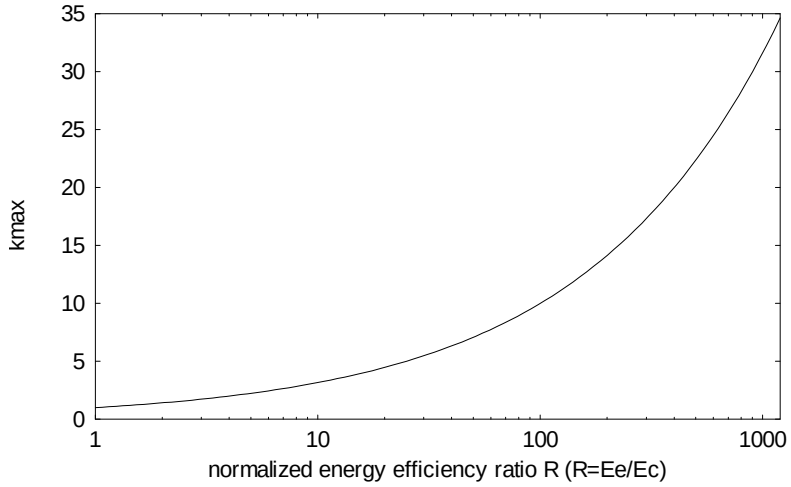


**Figure 20: The energy efficient value $k_{max}$ vs. energy efficiency ratio $R$.**

This relation is valid for any number of redundant packets, and is thus also usable for the Reed-Solomon coding with more than two redundant packets.

When implementing an energy efficient error correction mechanism, the system must be made such that it can adapt its strategy according to the conditions of the wireless channel, the required Quality of Service, and the energy consumption needed. There are several parameters that must be taken into account.

- *Ratio R*. The ratio $R$ between the energy efficiency factor of error correction and the energy efficiency factor of communication is not dependent on the environment and the current state of the wireless channel. It is static for a certain system configuration and can be determined when designing or configuring the system.
- *Bad state probability e*. This probability is highly dependent on the current situation of the wireless channel. The parameter determines the required coding mechanism, the number of source packets $k$, and the number of redundant packets $n$-$k$. Major changes can be expected when changing to a complete different infrastructure.
- *Maximum burst error-size*. This parameter is also dependent on the current situation and mainly influences the required buffer space, but also influences the error correcting capabilities of the mechanism. If the required QoS can sustain the delays that are introduced with the buffer, then the size can be chosen large.

When all these parameters are known, then the error correction mechanism can be chosen such that it adheres to the required QoS (incorporating the error correcting capabilities and latency) at the most energy efficient way using the relation (22) and (26) described in this section.

# 6. Conclusion

In our study we have investigated the energy efficiency of error control schemes for the wireless channel, and concentrated on systems where Quality of Service provisioning is a major concern, like in wireless ATM systems. Since high error rates are inevitable to the wireless environment, *energy efficient error control* is an important issue for mobile computing systems. Although good designed retransmission schemes can be optimal with respect to energy efficiency, they can introduce intolerable low performance in delay, jitter and bandwidth to fulfill the required QoS of the application. Therefor we have concentrated on *error correction* mechanisms. The two main external and dynamic parameters involved are burst-size and bad-state probability.

Error control mechanisms traditionally trades off complexity and buffering requirements for throughput and delay. The main design criteria is traditionally on optimizing the performance and providing an optimal solution to withstand all possible errors. In the context of energy constraints, error control mechanisms show a trade-off between added complexity and computation and increased communication. In our approach we apply energy consumption constraints to the error control mechanisms in order to enhance energy efficiency under various wireless channel conditions. We have shown that it is not sufficient to concentrate on the efficiency of error control mechanisms only, and that the energy efficiency trade-off between error correction and communication has a great impact on the energy efficiency of the system.

Error correcting is an area that is quite well suited for reconfiguration because (1) there are many error correcting algorithms possible, each offering a considerable amount of variability in the parameters and (2) because it can be adapted to a changed state of the wireless channel. Using *adaptable error correction*, it is possible to balance on the optimum trade-off between performance and cost. Environment changes are due to physical impairments and user movement. The required error correcting capability is determined by the required QoS of the application. A general error control scheme that operates *independently* most of the time either wastes energy for computation and communication because it provides a too high error correcting capability, or saves energy but provides an intolerable bad error correction. The most appropriate place to have error control is near the application and the QoS manager of the system. The QoS and thus the required error correction is determined by application, and thus the application needs to be able to influence the error control mechanism to use. An optimal and suitable error control scheme must be provided on a per-application basis. It must not be incorporated in some hidden data link layer, but near to the application.

We have studied three different error correction mechanisms with different characteristics and capabilities, i.e. simple parity, EVENODD, and Reed-Solomon. The implementations of these mechanisms on a general-purpose processor in *C* show that they already reach constant performance and constant energy efficiency for small values of $k$ and for small data sizes. The Reed-Solomon code is attractive because it is the most general technique capable of tolerating $n$-$k$ simultaneous failures. The complexity and the requirement of computations in the finite field make the encoding however about four times less energy efficient than other techniques like EVENODD and Parity. Decoding is more than two times as inefficient. We have shown that although the EVENODD scheme was originally designed for RAID systems with a number of disks, it can also very well be applied in communication systems. The EVENODD mechanism is based on exclusive-OR operations only, but can only sustain two packet erasures. It however allows the reconstruction of erased *symbols* (and not packet erasures as with Reed-Solomon) which increases its flexibility and gives a further reduction in energy consumption for decoding when the burst-error size is small. The results of Parity mechanism show, although it has a high efficiency, that it is only little more efficient than EVENODD. Since it is only an ($k$+1,$k$) code, it provides less error correction capability.

We have provided a strategy that can be used to determine the most energy efficient error correction scheme of a small *system* consisting of a general-purpose processor and a wireless communication interface. It was shown that from energy efficiency perspective it is not always profitable to minimize the number of bits transmitted over the air. There exist an optimum value for $k$ that should be used to have a suitable fault tolerance at the most energy efficient way. The error correction code is then

determined by two values of $k$: $k_{max}$ (for energy efficiency) and $k_{min}$ (for error correcting capabilities). As an example we have shown the energy efficiency of systems with the WaveLAN wireless interface.

*General conclusion*

Error control schemes for systems where Quality of Service provisioning is a major concern often cannot use retransmission schemes and need to use error correction mechanisms. When error correction mechanisms are being used for wireless systems a major design criterion should be the energy efficiency of a mechanism. *Adaptable error correction*, that adapts its parameters and scheme according to the error-rate and required QoS, can be used to trade-off between performance and cost, including the required energy consumption.

It is not sufficient to concentrate on the efficiency of error control mechanisms only, but the required extra energy consumed by the wireless interface should be incorporated as well. From energy efficiency perspective it is not always profitable to minimize the number of bits transmitted over the air. We have provided a strategy to determine the most energy efficient error correction scheme of a small *system* consisting of a general-purpose processor and a wireless communication interface.

# 7. References


[Awater 97] Awater, G, personal communication.

[Balakrishnan 96] Balakrishnan H., et al.: "A comparison of mechanisms for improving TCP performance over wireless links", Proceedings ACM SIGCOMM'96, Stanford, CA, USA, August 1996.

[Berlekamp 68] Berlekamp, E.R.,: "Algebraic Coding Theory", McGraw-Hill, 1968

[Birk 98] Y. Birk and Y. Keren,: "Judicious Use of Redundant Transmissions in Multi-Channel ALOHA Networks with Deadlines", proceedings IEEE Infocom'98, pp. 332-338, March 1998.

[Blaum 95] Blaum M., et al.: "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures", IEEE Transactions on computers, Vol. 44, No 2, pp. 192-201, February 1995.

[Borriss 95] Borriss, M. "QoS support in ATM and selected protocol implementations", technical report, TU Dresden, IBDR, http://www.inf.tu-dresden.de/~mb14/atm.html.

[Cho 94] Cho, Y.J., Un, C.K.: "Performance analysis of ARQ error controls under Markovian block error pattern", IEEE Transactions on Communications., Vol. COM-42, pp. 2051-2061, Feb-Apr. 1994.

[Chockalingam 98] Chockalingam, A., Zorzi, M.: "Energy consumption performance of a class of access protocols for mobile data networks", VTC'98, Ottawa, Canada, May 1998.

[Elaoud 98] Elaoud, M, Ramanathan, P: "Adaptive Use of Error-Correcting Codes for Real-time Communication in Wireless Networks", proceedings IEEE Infocom'98, pp. 548-555, March 1998.

[Ferrari 92] Ferrari, D.: "Real-Time Communication in an Internetwork", Journal of High Speed Networks, Vol. 1, n. 1, pp. 79-103, 1992

[Figueira 98] Figueira, N.R., Pasquale, J.: "Remote-Queueing Multiple Access (RQMA): Providing Quality of Service for Wireless Communications", proceedings IEEE Infocom'98, pp. 307-314, March 1998.

[Hartman 93] Hartman, J.H., Ousterhout, J.K.: "The zebra striped network file system", Operating Systems Review, 14th ACM symposium on Operating Systems Principles, 27(5):29-43, December 1993.

[Havinga 97] Havinga, P.J.M., Smit, G.J.M.: "Minimizing energy consumption for wireless computers in Moby Dick", proceedings IEEE International Conference on Personal Wireless Communication ICPCW'97, Dec. 1997.

[Huitema 96] Huitema, C.: "The case for packet level FEC", Proceedings 5th workshop on protocols for high speed networks, pp. 109-120, Sophia Antipolis, France, Oct. 1996.

[Lin 84] Lin, S., Costello, D.J., Miller, M.: "Automatic-repeat-request error-control schemes", IEEE Comm. Magazine, v.22, n.12, pp. 5-17, Dec 1984.

[Liu 96] Liu, H., El Zarki, M.: "Delay bounded type-II hybrid ARQ for video transmission over wireless networks", proceedings Conference on Information Sciences and Systems, Princeton, March 1996.

[Lorch 95] Lorch, J.: "A complete picture of the energy consumption of a portable computer", Masters Thesis, Computer Science, University of California at Berkeley, December 1995.

[LSI logic] "L64711/12/13/14 Reed-Solomon Encoders/decoders", LSI logic, http://www.lsilogic.com/products/unit5_7d.html.

[MacWilliams 77] MacWilliams, F.J., Sloane, N.J.A.: "The theory of error-correcting codes", North-Holland Publicing Company, Amsterdam, 1977.

[Mathis 96] Mathis, M., et al., "RFC2018: TCP selective acknowledgement option", Oct. 1996.

[Nonnenmacher 96] Nonnenmacher, J., Biersack, E.W.: "Reliable multicast: where to use Forward Error Correction", Proceedings 5th workshop on protocols for high speed networks, pp. 134-148, Sophia Antipolis, France, Oct. 1996.

[Patterson 88] Patterson, D. A., Gibson, G. A., Katz, R.: “A case for redundant arrays of inexpensive disks”, proceedings SIGMOD/ACM Int. Conference Data Management, pp. 109-116, 1988.

[PentiumPro] “Pentium Pro processors, Product overview”, http://developer.intel.com/design/pro.

[Plank 97] Plank, J.S.: “A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems”, Software, practice & experience”, 27(9), Sept 1997, pp. 995-1012.

[Prycker 94] De Prycker,M: “Asynchronous Transfer Mode”, 2nd. ed.; 1994 Ellis Horwood

[Rizzo 97] Rizzo, L.: “Effective Erasure Codes for Reliable Computer Communication Protocols”, ACM Computer Communication Review, Vol. 27- 2, pp 24-36, April 97.

[Rizzo 98] Rizzo, L., sources for an erasure code based on Reed-Solomon coding with Vandermonde matrices. Available at http://www.iet.unipi.it/~luigi/vdm.tgz.

[SA-1100] “StrongARM products, SA-1100 Microprocessor information Sheet”, http://developer.intel.com/design/strong/sa1100.htm.

[Shacham 90] Shacham, N., McKenney, P.: “Packet recovery in high-speed networks using coding and buffer management”, Proceedings IEEE Infocom’90, San Fransisco, pp. 124-131, May 1990.

[Smit 98] Smit, G.J.M., et al.: “Overview of the Moby Dick project”, Euromicro summer school on mobile computing, Oulu, Aug. 1998.

[Stemm 96] Stemm, M, et al.: “Reducing power consumption of network interfaces in hand-held devices”, Proceedings mobile multimedia computing MoMuc-3, Princeton, Sept 1996.

[WaveLAN] “WaveLAN/PCMCIA network adapter card”, http://www.wavelan.com/support/libpdf/fs-pcm.pdf.

[RS-Xilinx] “Xilinx Ships Industry's First Complete Programmable Logic Reed-Solomon Solution from AllianceCORE Program”, http://www.oakridge.com/xilinx/reedsolomon.html, or http://www.xilinx.com/products/logicore/tbl_comm_net.htm#fec.

[Zorzi 96] Zorzi, M., Rao, R. R.: “On the statistics of block errors in bursty channels”, IEEE transactions on communications

[Zorzi 97] Zorzi, M., Rao, R. R.: “Error control and energy consumption in communications for nomadic computing”, IEEE transactions on computers, Vol. 46, pp. 279-289, March 1997.

[Zorzi 98] Zorzi, M: “Performance of FEC and ARQ Error control in bursty channels under delay constraints”, VTC’98, Ottawa, Canada, May 1998.