Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
An Institute of National Importance
www.iiitdm.ac.in
COM302T Computer Networks-Lecture Notes

Instructor
N.Sadagopan

Error Detection and Correction

Objective: In this lecture, we shall explore various error detection techniques and its limitations. Further, we shall be discussing error correction techniques, and the amount of overhead due to detection and correction schemes.

Introduction

A bit, on travel, is subjected to electromagnetic (optical) interference due to noise signals (light sources). Thus, the data transmitted may be prone to errors. In particular, a bit '0' (bit '1') sent by the sender may be delivered as a bit '1' (bit '0') at the receiver. This happens because the voltage present in noise signals either has direct impact on the voltage present in the data signal or creates a distortion leading to mis-interpretation of bits while decoding the signals at the receiver. This calls for a study on error detection and error correction. The receiver must be intelligent enough to detect the error and ask the sender to transmit the data packet or must have the ability to detect and correct the errors.

Usually, noise signals if it all occurs, stay with the data signals for a longer duration and corrupts (changes bit '1' to bit '0' or bit '0' to bit '1') multiple bits. For example, on a 100 Mbps link, if the noise signal stays for a 10 micro sec, then it is likely to corrupt $100 \times 10^6 \times 10 \times 10^{-6} = 1000$ bits. Similarly, to corrupt a single bit on a 100 Mbps link, the noise signal must stay active for 0.01 micro sec.
In this lecture, we shall discuss the following error detection techniques in detail.

1. Vertical Redundancy check
2. Longitudinal Redundancy check
3. Checksum
4. Cyclic Redundancy check

Vertical Redundancy Check (VRC)

This error detection scheme is quite popular in telecommunications and computer networking. The scheme works as follows: the message to be transmitted is split into nibbles (4 bits) and with each nibble a parity bit is associated before data transmission. Note that with even parity bit scheme, the number of 1's in the nibble including the parity bit must be even. For example, for the message 1 0 0 1 0 0 0 1 1 1 1 1 with even parity bit scheme, the message to be transmitted is

1 0 0 1 $\boxed{0}$ 0 0 0 1 $\boxed{1}$ 1 1 1 1 $\boxed{0}$

The bit inside the 'box' at the end of the nibble represents the even parity bit corresponding to that nibble.

Remarks:

1. This scheme detects all single bit errors. Further, it detects all multiple bit errors as along as the number of bits corrupted is odd (referred to as odd bit errors).
   Suppose the message to be transmitted is

   $$1\ 0\ 1\ 1\ \boxed{1} \qquad 1\ 0\ 0\ 0\ \boxed{1} \qquad 1\ 0\ 0\ 1\ \boxed{0}$$

   and the message received at the receiver is

   $$1\ 0\ \hat{0}\ 1\ \boxed{1} \qquad 1\ 0\ 0\ 0\ \boxed{1} \qquad 1\ 0\ 0\ 1\ \boxed{0}$$

2. $\hat{0}$ represents that this bit is in error. For the above example, when the receiver performs the parity check, it detects that there was an error in the first nibble during transmission as there is a mismatch between the parity bit and the data in the nibble. However, the receiver does not know which bit in that nibble is in error. Similarly, the following error is also detected by the receiver.
   the message to be transmitted is

   $$1\ 0\ 1\ 1\ \boxed{1} \qquad 1\ 0\ 0\ 0\ \boxed{1} \qquad 1\ 0\ 0\ 1\ \boxed{0}$$

   and the message received at the receiver is

   $$\hat{0}\ \hat{1}\ 1\ 1\ \boxed{1} \qquad 1\ 0\ 0\ \hat{1}\ \boxed{1} \qquad 1\ 0\ 0\ 1\ \boxed{0}$$

   Three bits are corrupted by the transmission medium and this is detected by the receiver as there is a mismatch between the 2^{nd} nibble and the 2^{nd} parity bit. It is important to note that the error in the first nibble is unnoticed as there is no mismatch between the data and the parity bit.

3. The above example also suggests that not all even bit errors (multiple bit errors with the number of bits corrupted being even) are detected by this scheme. If even bit error is such that the even number of bits are corrupted in each nibble, then such error is unnoticed at the receiver. However, if even bit error is such that at least one of the nibble has odd number of bits in error, then such error is detected by VRC.

4. If nibble based VRC scheme is followed, then the overhead is 25%, i.e., if the size of the message is m, the number of even parity bits is $\frac{m}{4}$, which is 25% overhead. In general, if the size of the split is k, then the number of even parity bits is $\frac{m}{k}$ and the percentage of overhead is $\frac{100}{k}$. It is easy to see that the higher the value of k, the lesser the percentage of overhead. However, there is a trade-off between k and the error detection efficiency. The error detection efficiency refers to the percentage of errors being detected by VRC when simulated on a large number of messages. In general, the lower the value of k, the higher the error detection efficiency.

Longitudinal Redundancy Check (LRC)

In contrast to VRC, LRC assigns a parity byte (nibble) along with the message to be transmitted. Suppose the message to be transmitted is

$$1\ 0\ 1\ 1 \qquad 1\ 0\ 0\ 0 \qquad 1\ 0\ 0\ 1$$

then, we compute the even parity nibble as follows;

$$\begin{array}{c} 1\ 0\ 1\ 1 \\ 1\ 0\ 0\ 0 \\ 1\ 0\ 0\ 1 \\ \boxed{1\ 0\ 1\ 0} \end{array}$$

Note that in this scheme, the number of 1's in each column including the bit in the parity nibble must be even.

Remarks:

1. Similar to VRC, LRC detects all single bit and odd bit errors. Some even bit errors are detected and the rest is unnoticed by the receiver.

2. The following error is detected by LRC but not by VRC. For the message 1 0 1 1 1 0 0 0 1 0 0 1 , suppose the received message is

$$\begin{array}{c} 1\ \hat{1}\ \hat{0}\ 1 \\ 1\ 0\ 0\ 0 \\ 1\ 0\ 0\ 1 \\ \boxed{1\ 0\ 1\ 0} \end{array}$$

3. In the above example, there is mismatch between the number of 1's and the parity bit in Columns 2 and 3.

4. The following error is detected by VRC but not by LRC. For the message 1 0 1 1 1 0 0 0 1 0 0 1 , suppose the received message is

$$\begin{array}{c} 1\ 0\ 1\ \hat{0} \\ 1\ 0\ 0\ \hat{1} \\ 1\ 0\ 0\ 1 \\ \boxed{1\ 0\ 1\ 0} \end{array}$$

   The above error is unnoticed by the receiver if we follow VRC scheme whereas it is detected by LRC as illustrated below.

$$1\ 0\ 1\ \hat{0}\ \boxed{1} \qquad 1\ 0\ 0\ \hat{1}\ \boxed{1} \qquad 1\ 0\ 0\ 1\ \boxed{0}$$

5. Here again, not all even bit errors are detected by this scheme. If the error is such that each column has even number of bits in error, then such error is undetected. However, if the distribution is such that at least one column contains an odd number of bits in error, then such errors are always detected at the receiver.

Checksum

This scheme is a peculiar one wherein we perform 1's complement arithmetic on the data to be transmitted. For the message 1 0 1 1 1 0 0 0 1 0 0 1 , we compute `checksum` in two steps. The first step performs 1's complement addition on the data and the second step performs the complement on the result of Step 1. The result of Step 2 is the checksum which would be augmented along with the data to be transmitted.

Step: 1 Perform 1's complement addition

$$
\begin{array}{r}
1\ 0\ 1\ 1 \\
1\ 0\ 0\ 0 \\
1\ 0\ 0\ 1 \\
\hline
1\ 1\ 0\ 0 \\
\qquad 1 \qquad \text{(carry from the addition)} \\
\hline
1\ 1\ 0\ 1
\end{array}
$$

Step: 2 Compute Checksum which is the 1's complement of the result of addition.

For the above example, the checksum is 0 0 1 0. Along with the message, we append the checksum of the message. i.e., the message to be transmitted is

$$1\ 0\ 1\ 1 \quad 1\ 0\ 0\ 0 \quad 1\ 0\ 0\ 1 \quad \boxed{0\ 0\ 1\ 0}$$

At the receiver, we perform 1's complement addition on the received data (message plus checksum), if the result is 1 1 1 1, then the receiver declares that there is no error in the transmission. Otherwise, it declares that there is an error in the transmission.

It is now natural to ask the efficiency of this scheme. Does checksum outperform VRC/LRC ? Does checksum detect all errors?
The following error is detected by checksum but not by LRC. For the message $1\ 0\ 1\ 1 \quad 1\ 0\ 0\ 0 \quad 1\ 0\ 0\ 1 \quad \boxed{0\ 0\ 1\ 0}$, suppose the received message is

$$
\begin{array}{l}
\hat{0}\ 0\ 1\ 1 \\
\hat{0}\ 0\ 0\ 0 \\
1\ 0\ 0\ 1 \\
\boxed{0\ 0\ 1\ 0}
\end{array}
$$

On performing 1's complement arithmetic at the receiver, we see that the result is not 1 1 1 1, and therefore, there is an error in transmission.

$$
\begin{array}{l}
\hat{0}\ 0\ 1\ 1 \\
\hat{0}\ 0\ 0\ 0 \\
1\ 0\ 0\ 1 \\
\boxed{0\ 0\ 1\ 0} \\
\hline
1\ 1\ 1\ 0
\end{array}
$$

This error is unnoticed by LRC scheme as even number of identical bits are corrupted in a column and this change has no impact on the even parity bit. However, checksum performs 1's complement addition and hence there will be a change in the final sum if even number of identical bits are corrupted.
The following error is unnoticed by checksum and also by LRC. For the message $1\ 0\ 1\ 1 \quad 1\ 0\ 0\ 0 \quad 1\ 0\ 0\ 1$, suppose the received message is

$$
\begin{array}{l}
1\ 0\ 1\ \hat{0} \\
1\ 0\ 0\ \hat{1} \\
1\ 0\ 0\ 1 \\
\boxed{0\ 0\ 1\ 0}
\end{array}
$$

On performing 1's complement arithmetic at the receiver, we get

$$
\begin{array}{r}
1\ 0\ 1\ \hat{0} \\
1\ 0\ 0\ \hat{1} \\
1\ 0\ 0\ 1 \\
\boxed{0\ 0\ 1\ 0} \\
\hline
1\ 1\ 1\ 0 \\
\qquad 1 \quad \text{(carry from the addition)} \\
\hline
1\ 1\ 1\ 1
\end{array}
$$

Thus, the receiver incorrectly declares that there is no error in the transmission. This is unnoticed as the sum is unaffected due to the fact the bits in error are complement to each other.
Remarks:

1. If multiple bit error is such that in each column, a bit '0' is flipped to bit '1', then such an error is undetected by this scheme. Essentially, the message received at the receiver has lost the value 1 1 1 1 with respect to the sum. Although, it loses this value, this error is unnoticed at the receiver. Why?
2. Also, multiple bit error is such that the difference between the sum of the sender's data and the sum of receiver's data is 1 1 1 1, then this error is unnoticed by the receiver. Why?
3. The above two errors are undetected by the receiver due to the following interesting observations.
4. For any binary data a such that $a \neq 0\ 0\ 0\ 0$, the value of $a + (1\ 1\ 1\ 1)$ in 1's complement arithmetic is a. On the similar line, if $a_1, \ldots, a_k$ are nibbles, then $a_1 + \ldots + a_k + (1\ 1\ 1\ 1) = a_1 + \ldots + a_k$. This is true because, $a + 1\ 1\ 1\ 1$ gives $a - 1$ with a carry '1' and inturn this '1' is added with $a - 1$ as part of 1's complement addition, yielding a.
5. Consider the scenario in which the sender transmits $a_1, \ldots, a_k$ along with the checksum and the transmission line corrupts multiple bits due to which we lose 1 1 1 1 on the sum. Although, the receiver receives $a_1 + \ldots + a_k - (1\ 1\ 1\ 1)$, it follows from the above observation that $a_1 + \ldots + a_k - (1\ 1\ 1\ 1)$ is still $a_1 + \ldots + a_k$, thus the error is undetected.
6. Similar to other error detection schemes, checksum detects all odd bit errors and most of even bit errors.

Cyclic Redundancy Check (CRC)

CRC performs mod 2 arithmetic (exclusive-OR) on the message using a divisor polynomial. Firstly, the message to be transmitted is appended with CRC bits and the number of such bits is the degree of the divisor polynomial. The divisor polynomial 1 1 0 1 corresponds to the polynomial $x^3 + x^2 + 1$. For example, for the message 1 0 0 1 0 0 with the divisor polynomial 1 1 0 1, the message after appending CRC bits is 1 0 0 1 0 0 0 0 0. We compute CRC on the modified message M as follows;

How to compute CRC bits:
1. To perform xor arithmetic, the leading bit of M must be '1'. If leading bit is not '1', then choose the first bit of M for which the bit value is '1'. Assuming the leading bit is '1', we perform xor arithmetic with the first $d + 1$ bits of M. This results in a value with the leading bit '0'. If suppose the next bit of the leading bit is '1', then we bring the next bit ($(d + 2)^{nd}$ bit) from M so that we will have $d + 1$ bits with a leading bit '1' for the next xor operation. Otherwise,

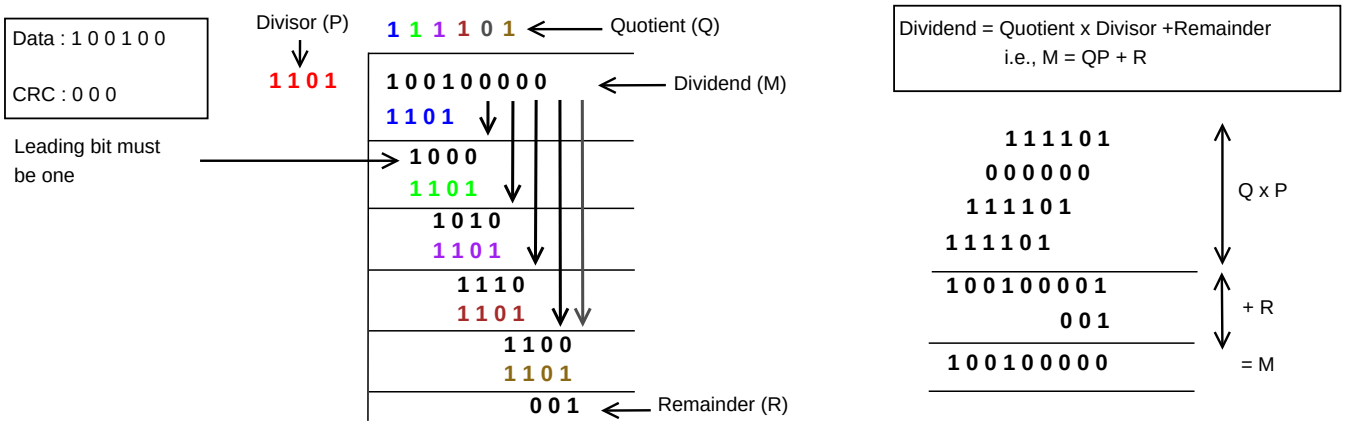

Fig. 1. CRC Computation

we identify the first bit in the result which is '1' and bring appropriate number of bits from M so that we will have $d+1$ bits with a leading bit '1' for the next xor operation. At each step of computation, we ensure the leading bit is '1' and perform xor arithmetic on $d+1$ bits of M with the divisor polynomial.

2. For the example illustrated in Figure 1, we see that the remainder is 0 0 1 which is the CRC bits. Further, CRC bits is appended with the message to be transmitted (equivalent to performing xor between CRC and the modified message) before data transmission. Let M' be the message to be transmitted by the sender.

3. At the receiver, on receiving the message M', the receiver performs CRC check similar to the sender. That is, M' xor divisor polynomial is performed to check whether there is an error in data transmission. If the remainder of the CRC check is zero, then the receiver declares that there is no error, otherwise, it declares that there is an error in transmission. It is now natural to ask the efficiency of CRC scheme. Does CRC outperform all other error detection schemes? Does CRC detect all multiple bit errors?

Proof of correctness of CRC scheme:

We shall now discuss why xor arithmetic works correctly when there is no error in the data transmission. I.e., when there is no error in the transmission and the receiver gets the remainder zero, then this scheme says, the message contained in the received data is without error. Let us fix some notation to outline the proof of correctness. Let M be the message to be transmitted and n be the number of CRC bits (the degree of CRC polynomial).

- The sequence of steps performed at the sender is as follows; appending CRC bits is equivalent to shifting M left by n positions. The decimal equivalent of the modified message is 2^nM. Subsequently, we perform CRC computation on the modified message and append CRC bits (i.e., xor the remainder R) with the modified message, which is precisely 2^nM+R. Note that '+' in the summand denotes xor addition.

- It is important to note that $2^nM = PD + R$ where P is the quotient, D is the divisor, and R is the remainder upon dividing 2^nM with D.

– The message to be transmitted is 2^nM+R and on receiving 2^nM+R, the receiver performs similar CRC computation. That is, it computes $\frac{2^nM+R}{D}$. On simplifying, we get, $\frac{2^nM}{D}+\frac{R}{D}$. Since $2^nM = PD+R$, $\frac{PD+R}{D}+\frac{R}{D} = P+\frac{R}{D}+\frac{R}{D}$. Since the addition is xor, $\frac{R}{D}+\frac{R}{D}=0$. Thus, we get P with the remainder zero. This shows that the CRC computation works perfect when there is no error in data transmission. Further, it is appropriate to conclude that there is no error in transmission when CRC computation leaves the remainder zero and there is an error in transmission when CRC computation leaves the remainder non-zero.

Does CRC detect all errors?

– The answer to this question depends on the divisor polynomial used as part of CRC computation. Consider the divisor polynomial x^2 which corresponds to 1 0 0 and the message to be transmitted is 1 0 1 0 1 0. After appending CRC bits (in this case 0 0) we get 1 0 1 0 1 0 0 0. Assuming during the data transmission, the 3^{rd} bit is in error and the message received at the receiver is 1 0 1 0 1 $\hat{1}$ 0 0. On performing CRC check on 1 0 1 0 1 1 0 0, we see that the remainder is zero and the receiver wrongly concludes that there is no error in transmission. The reason this error is undetected by the receiver is that the message received is perfectly divisible by the divisor 1 0 0. More appropriately, if we visualize the received message as the xor of the original message and the error polynomial, then we see that the error polynomial is divisible by 1 0 0.

– Message received = 1 0 1 0 1 1 0 0
Message received = message transmitted + error polynomial, i.e.,

1 0 1 0 1 1 0 0= 1 0 1 0 1 0 0 0 + 0 0 0 0 0 1 0 0.

Clearly both are divisible by the divisor 1 0 0. In general, if the error polynomial is divisible by the divisor, then such errors are undetected by the receiver. The receiver wrongly concludes that there is no error in transmission. For the example, if the error polynomial is 0 0 0 0 1 1 0 0 (3^{rd} and 4^{th} bits in the message in error), then the receiver is unaware of this error. However, if the divisor is 1 0 1, then both errors are detected by the receiver. Thus, choosing an appropriate divisor is crucial in detecting errors at the receiver if any during the transmission.

– We below mention a few divisor polynomials and error polynomials which are divisible by divisor polynomials.

Divisor Polynomial	Error Polynomials	Remarks
101	101, 10001, 110011	101 detects all single bit errors and does not detect all double bit errors.
1011	10000001	does not detect all double bit errors, detects all single bit errors.
1101	10000001	does not detect all double bit errors, detects all single bit errors.

– In general, the divisor polynomials are such that it detects almost all odd bit errors and a few even bit errors. The commonly used polynomials in the literature are
CRC-8: $x^8 + x^2 + x + 1$
CRC-16: $x^{16} + x^{15} + x^2 + 1$
CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

– CRC-32 is such that it detects all multiple bit errors of size at most 31 and multiple bit errors of size at least 32 is detected with high probability.

Recap: We have discussed various error detection schemes such as VRC, LRC, checksum and CRC, and their limitations. Interestingly, no error detection scheme is perfect and this leaves open the direction for further research in finding a perfect error detection scheme. We shall discuss next a scheme which detects and corrects all single bit errors. That is the receiver is intelligent enough to identify the bit in error so that it can be flipped to get the actual message.

Hamming Error Correcting Code

In this section, we shall discuss an error detection and a correction scheme to handle single bit errors. This scheme is due to R.Hamming, a well-known researcher in the field of telecommunications. Like any other scheme, Hamming Error Correcting Code (HEC) augments redundancy bits using which it identifies which bit is in error. We shall determine the number of redundancy bits r required for a message of size m as follows;

Note that the message to be transmitted along with the redundancy bits is $m + r$ and the receiver can be in one of the following states. (i) No error, i.e., none of $m + r$ bits is in error (ii) the first bit is in error (iii) the second bit is in error and so on. This implies that

$2^r \geq (m+r)+1$, where 2^r denotes the number of possible configurations of redundancy bits and each configuration represents a state of the receiver, and the number of such states is $(m+r)+1$ (any one of the $m + r$ bits in error or no error).

For $m = 10$, to detect all single bit errors, we need $r = 4$ bits and for a message $m = 1000$, we need $r = 10$ bits.
We shall now discuss HEC using which we can determine the value of redundancy bits. HEC follows even parity scheme. Suppose the message to be transmitted is 1 0 0 1 1 0 1 0. Since $m = 8$, $r = 4$. We associate four redundancy bits r_1, r_2, r_4, r_8 which we compute using the values present in the following bit positions. The value of r_1 is '1' if the number of 1's in the following bit positions including the bit r_1 is even. Similarly, for r_2 and other redundancy bits.
$r_1 = 1, 3, 5, 7, 9, 11$
$r_2 = 2, 3, 6, 7, 10, 11$
$r_4 = 4, 5, 6, 7, 12$
$r_8 = 8, 9, 10, 11, 12$

Bit Positions	12	11	10	9	8	7	6	5	4	3	2	1
Message	1	0	0	1		1	0	1		0		

For the above message the values of $r_1 = 1$, $r_2 = 1$, $r_4 = 1$ and $r_8 = 0$.

The message to be transmitted is 1 0 0 1 $\boxed{0}$ 1 0 1 $\boxed{1}$ 0 $\boxed{1}$ $\boxed{1}$
On receiving the message, the receiver computes the values of four redundancy bits r_1, r_2, r_4, r_8

to ascertain whether there was an error in the data transmission and these bits also help us in identifying the bit in error.

- Suppose there is no error in transmission. Then the received message is 1 0 0 1 $\boxed{0}$ 1 0 1 $\boxed{1}$ 0 $\boxed{1}$ $\boxed{1}$. It is easy to see that $r_1 = 0$, $r_2 = 0$, $r_4 = 0$ and $r_8 = 0$. This corresponds to the decimal value '0' which means there is no error in the transmission.
- Suppose 3^{rd} bit is in error. Then the received message is 1 0 0 1 $\boxed{0}$ 1 0 1 $\boxed{1}$ $\hat{\mathbf{1}}$ $\boxed{1}$ $\boxed{1}$. On computing redundancy bits, we get $r_1 = 1$, $r_2 = 1$, $r_4 = 0$ and $r_8 = 0$. This corresponds to the decimal value '3' which means 3^{rd} bit is in error and the receiver flips the third bit to get the actual message.

Why HEC works: The answer to this question lies in the bit positions chosen by HEC. For the above example, the total size of the message including the redundancy bits is 12. Consider the first 12 decimal numbers starting from 1 and its equivalent binary value in 8 4 2 1 representation. It is important to note that with respect to 8 4 2 1, the bit positions mentioned in r_1 are those where the first position is one, r_2 corresponds to the positions where the second bit is one, and so on.
Now, during the transmission, if the 3^{rd} bit is in error, then the value of r_1 and r_2 will be '1' and the other two will be '0' when the computation is done at the receiver. As the decimal equivalent is three as per 8421 encoding, the receiver knows that it is 3^{rd} bit which is in error.
Suppose $m = 17$, then $r = 5$. In such a case, we shall work with 16 8 4 2 1 encoding scheme and list the binary equivalent of the decimals 1 to 22. The bit r_1 corresponds to the bit positions where the first bit is '1'.

Redundancy bits for Multiple bit errors:
As mentioned before, to detect all single bit errors on a message of size m with r redundancy bits, the number of states is given by $2^r \geq (m+r)+1$. On the similar line, to detect all burst errors of size at most two, the number of states in which the receiver can be in is $2^r \geq 1+(m+r)+\binom{(m+r)}{2}$. In general, to detect all burst errors of size at most k, the number of states that the receiver can be in is $2^r \geq 1 + (m + r) + \binom{(m+r)}{2} + \ldots + \binom{(m+r)}{k}$.

Two Dimensional Parity Scheme

In this scheme, with the message to be transmitted, we augment a parity bit and a parity byte. In some sense, it is a blend of VRC and LRC.

1 0 1 1	$\boxed{1}$
1 0 0 0	$\boxed{0}$
1 0 0 1	$\boxed{0}$
$\boxed{1\ 0\ 1\ 0}$	$\boxed{1}$

Interestingly, this scheme detects and corrects all single bit errors. Suppose for the above message, the received message is

1 0 1 $\hat{1}$	$\boxed{1}$
1 0 0 0	$\boxed{0}$
1 0 0 1	$\boxed{0}$
$\boxed{1\ 0\ 1\ 0}$	$\boxed{1}$

On computing parity byte and bit at the receiver, the receivers finds a mis-match at Row 1 and Column 4, and this implies that the entry $(1, 4)$ is in error. By flipping the bit in $(1, 4)$, the receiver can get the original message. In general, when there is a single bit error at position (x, y), the receiver detects the error as there will be mismatch at Row x and Column y. As far as multiple bit errors are concerned, odd bit errors are detected (only detection, correction may not be possible) and some even bit errors are detected.
The following double bit error is detected, whereas the correction is not possible as there is some uncertainty in identifying the location of bits.

$$\begin{array}{|cccc|c|}\hline 1 & \hat{1} & 1 & 1 & 1 \\ \hat{0} & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 0 & \\ \hline & & & & 1 \\ \hline \end{array}$$

The receiver on applying this scheme detects that there are errors in Row 1, Row 2, Column 1 and Column 2. However, the receiver is unable to decide which two of $(1, 1), (1, 2), (2, 1), (2, 2)$ are in error. Thus, the detection is possible whereas the correction is not possible due to the above ambiguity.
The following triple bit error is detected but not corrected by the receiver. Moreover, the below triple bit error may be recognized as a single bit error. The following triple bit error is misclassified as a single bit error. On performing parity check, the receiver would declare that the bit $(1, 1)$ is in error which is incorrect.

$$\begin{array}{|cccc|c|}\hline 1 & 0 & 1 & \hat{0} & 1 \\ \hat{0} & 0 & 0 & \hat{1} & 1 \\ 1 & 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 0 & \\ \hline & & & & 1 \\ \hline \end{array}$$

HEC vs Two-dimensional parity:
For a message of size 1000, the HEC uses 10 redundancy bits. If we follow nibble based two-dimensional parity, then there are 250 rows and 4 columns, this implies that the overhead is $250 + 4 + 1 = 255$ bits. The overhead of two-dimensional parity can be reduced by choosing the block size to be 16 or 32 instead of 4. In general, HEC has less overhead and outperforms two-dimensional parity scheme for correcting single bit errors.