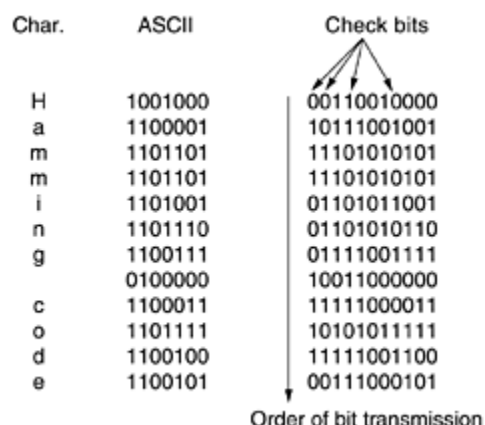When a codeword arrives, the receiver initializes a counter to zero. It then examines each check bit, k (k = 1, 2, 4, 8, ...), to see if it has the correct parity. If not, the receiver adds k to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the codeword is accepted as valid. If the counter is nonzero, it contains the number of the incorrect bit. For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only one checked by bits 1, 2, and 8. Figure 3-7 shows some 7-bit ASCII characters encoded as 11-bit codewords using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.

Figure 3-7. Use of a Hamming code to correct burst errors.

Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

Hamming codes can only correct single errors. However, there is a trick that can be used to permit Hamming codes to correct burst errors. A sequence of k consecutive codewords are arranged as a matrix, one codeword per row. Normally, the data would be transmitted one codeword at a time, from left to right. To correct burst errors, the data should be transmitted one column at a time, starting with the leftmost column. When all k bits have been sent, the second column is sent, and so on, as indicated in Fig. 3-7. When the frame arrives at the receiver, the matrix is reconstructed, one column at a time. If a burst error of length k occurs, at most 1 bit in each of the k codewords will have been affected, but the Hamming code can correct one error per codeword, so the entire block can be restored. This method uses kr check bits to make blocks of km data bits immune to a single burst error of length k or less.

3.2.2 Error-Detecting Codes

Error-correcting codes are widely used on wireless links, which are notoriously noisy and error prone when compared to copper wire or optical fibers. Without error-correcting codes, it would be hard to get anything through. However, over copper wire or fiber, the error rate is much lower, so error detection and retransmission is usually more efficient there for dealing with the occasional error.

As a simple example, consider a channel on which errors are isolated and the error rate is 10^{-6} per bit. Let the block size be 1000 bits. To provide error correction for 1000-bit blocks, 10 check bits are needed; a megabit of data would require 10,000 check bits. To merely detect a block with a single 1-bit error, one parity bit per block will suffice. Once every 1000 blocks, an extra block (1001 bits) will have to be transmitted. The total overhead for the error detection + retransmission method is only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

If a single parity bit is added to a block and the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5, which is hardly acceptable. The odds can be improved considerably if each block to be sent is regarded as a rectangular matrix n bits wide and k bits high, as described above. A parity bit is computed separately for each column and affixed to the matrix as the last row. The matrix is then transmitted one row at a time. When the block arrives, the receiver checks all the parity bits. If any one of them is wrong, the receiver requests a retransmission of the block. Additional retransmissions are requested as needed until an entire block is received without any parity errors.

This method can detect a single burst of length n, since only 1 bit per column will be changed. A burst of length n + 1 will pass undetected, however, if the first bit is inverted, the last bit is inverted, and all the other bits are correct. (A burst error does not imply that all the bits are wrong; it just implies that at least the first and last are wrong.) If the block is badly garbled by a long burst or by multiple shorter bursts, the probability that any of the n columns will have the correct parity, by accident, is 0.5, so the probability of a bad block being accepted when it should not be is 2^{-n}.

Although the above scheme may sometimes be adequate, in practice, another method is in widespread use: the polynomial code, also known as a CRC (Cyclic Redundancy Check). Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0. Such a polynomial is said to be of degree k - 1. The high-order (leftmost) bit is the coefficient of x^{k-1}; the next bit is the coefficient of x^{k-2}, and so on. For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1: $x^5 + x^4 + x^0$.

Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. There are no carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR. For example:

```
  10011011      00110011      11110000      01010101
+ 11001010    + 11001101    - 10100110    - 10101111
_____    _____    _____    _____
  01010001      11111110      01010110      11111010
```

Long division is carried out the same way as it is in binary except that the subtraction is done modulo 2, as above. A divisor is said "to go into" a dividend if the dividend has as many bits as the divisor.

When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, G(x), in advance. Both the high- and low-order bits of the generator must be 1. To compute the checksum for some frame with m bits, corresponding to the polynomial M(x), the frame must be longer than the generator polynomial. The idea is to append a checksum to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by G(x). When the receiver gets the checksummed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

The algorithm for computing the checksum is as follows:

1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m + r bits and corresponds to the polynomial $x^r M(x)$.
2. Divide the bit string corresponding to G(x) into the bit string corresponding to $x^r M(x)$, using modulo 2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).
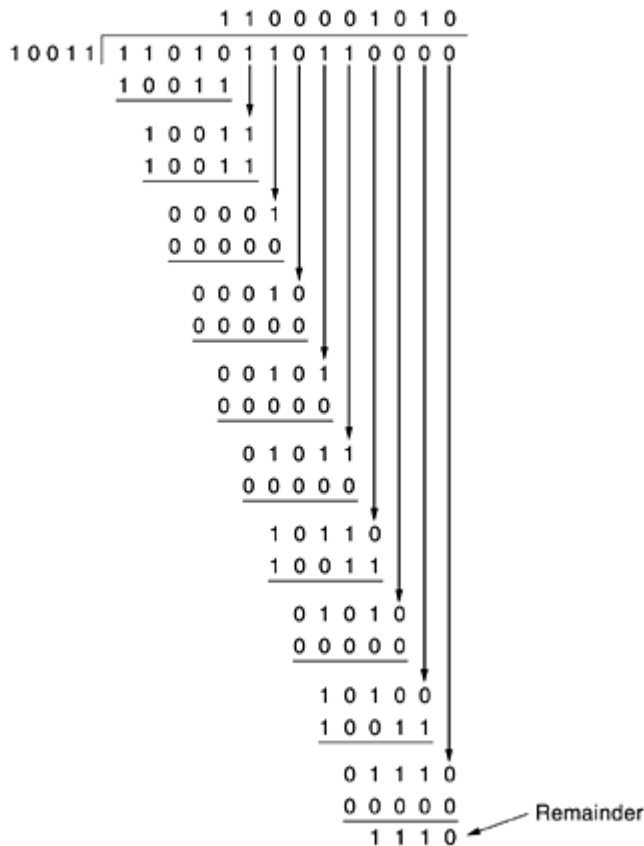
Figure 3-8 illustrates the calculation for a frame 1101011011 using the generator G(x) = x^4 + x + 1.

Figure 3-8. Calculation of the polynomial code checksum.

```
Frame    : 1 1 0 1 0 1 1 0 1 1
Generator: 1 0 0 1 1
Message after 4 zero bits are appended:  1 1 0 1 0 1 1 0 1 1 0 0 0 0

                              1 1 0 0 0 0 1 0 1 0
                     10011│ 1 1 0 1 0 1 1 0 1 1 0 0 0 0
                            1 0 0 1 1
                            ─────────
                              1 0 0 1 1
                              1 0 0 1 1
                              ─────────
                                0 0 0 0 1
                                0 0 0 0 0
                                ─────────
                                  0 0 0 1 0
                                  0 0 0 0 0
                                  ─────────
                                    0 0 1 0 1
                                    0 0 0 0 0
                                    ─────────
                                      0 1 0 1 1
                                      0 0 0 0 0
                                      ─────────
                                        1 0 1 1 0
                                        1 0 0 1 1
                                        ─────────
                                          0 1 0 1 0
                                          0 0 0 0 0
                                          ─────────
                                            1 0 1 0 0
                                            1 0 0 1 1
                                            ─────────
                                              0 1 1 1 0
                                              0 0 0 0 0      ── Remainder
                                              ─────────
                                                1 1 1 0 ◄──

Transmitted frame:  1 1 0 1 0 1 1 0 1 1 1 1 1 0
```

It should be clear that T(x) is divisible (modulo 2) by G(x). In any division problem, if you diminish the dividend by the remainder, what is left over is divisible by the divisor. For example, in base 10, if you divide 210,278 by 10,941, the remainder is 2399. By subtracting 2399 from 210,278, what is left over (207,879) is divisible by 10,941.

Now let us analyze the power of this method. What kinds of errors will be detected? Imagine that a transmission error occurs, so that instead of the bit string for T(x) arriving, T(x) + E(x) arrives. Each 1 bit in E(x) corresponds to a bit that has been inverted. If there are k 1 bits in E(x), k single-bit errors have occurred. A single burst error is characterized by an initial 1, a mixture of 0s and 1s, and a final 1, with all other bits being 0.

Upon receiving the checksummed frame, the receiver divides it by G(x); that is, it computes [T(x) + E(x)]/G(x). T(x)/G(x) is 0, so the result of the computation is simply E(x)/G(x). Those errors that happen to correspond to polynomials containing G(x) as a factor will slip by; all other errors will be caught.

If there has been a single-bit error, $E(x) = x^i$, where i determines which bit is in error. If G(x) contains two or more terms, it will never divide E(x), so all single-bit errors will be detected.

If there have been two isolated single-bit errors, $E(x) = x^i + x^j$, where i > j. Alternatively, this can be written as $E(x) = x^j(x^{i-j} + 1)$. If we assume that G(x) is not divisible by x, a sufficient condition for all double errors to be detected is that G(x) does not divide $x^k + 1$ for any k up to the maximum value of i - j (i.e., up to the maximum frame length). Simple, low-degree polynomials that give protection to long frames are known. For example, $x^{15} + x^{14} + 1$ will not divide $x^k + 1$ for any value of k below 32,768.

If there are an odd number of bits in error, E(X) contains an odd number of terms (e.g., $x^5 + x^2 + 1$, but not $x^2 + 1$). Interestingly, no polynomial with an odd number of terms has x + 1 as a factor in the modulo 2 system. By making x + 1 a factor of G(x), we can catch all errors consisting of an odd number of inverted bits.

To see that no polynomial with an odd number of terms is divisible by x + 1, assume that E(x) has an odd number of terms and is divisible by x + 1. Factor E(x) into (x + 1) Q(x). Now evaluate E(1) = (1 + 1)Q(1). Since 1 + 1 = 0 (modulo 2), E(1) must be zero. If E(x) has an odd number of terms, substituting 1 for x everywhere will always yield 1 as the result. Thus, no polynomial with an odd number of terms is divisible by x + 1.

Finally, and most importantly, a polynomial code with r check bits will detect all burst errors of length $\leq$ r. A burst error of length k can be represented by $x^i(x^{k-1} + ... + 1)$, where i determines how far from the right-hand end of the received frame the burst is located. If G(x) contains an x^0 term, it will not have x^i as a factor, so if the degree of the parenthesized expression is less than the degree of G(x), the remainder can never be zero.

If the burst length is r + 1, the remainder of the division by G(x) will be zero if and only if the burst is identical to G(x). By definition of a burst, the first and last bits must be 1, so whether it matches depends on the r - 1 intermediate bits. If all combinations are regarded as equally likely, the probability of such an incorrect frame being accepted as valid is $\frac{1}{2}^{r-1}$.

It can also be shown that when an error burst longer than r + 1 bits occurs or when several shorter bursts occur, the probability of a bad frame getting through unnoticed is $\frac{1}{2}^r$, assuming that all bit patterns are equally likely.

Certain polynomials have become international standards. The one used in IEEE 802 is

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

Among other desirable properties, it has the property that it detects all bursts of length 32 or less and all bursts affecting an odd number of bits.

Although the calculation required to compute the checksum may seem complicated, Peterson and Brown (1961) have shown that a simple shift register circuit can be constructed to compute and verify the checksums in hardware. In practice, this hardware is nearly always used. Virtually all LANs use it and point-to-point lines do, too, in some cases.

For decades, it has been assumed that frames to be checksummed contain random bits. All analyses of checksum algorithms have been made under this assumption. Inspection of real data has shown this assumption to be quite wrong. As a consequence, under some circumstances, undetected errors are much more common than had been previously thought (Partridge et al., 1995).

3.3 Elementary Data Link Protocols

To introduce the subject of protocols, we will begin by looking at three protocols of increasing complexity. For interested readers, a simulator for these and subsequent protocols is available via the Web (see the preface). Before we look at the protocols, it is useful to make explicit some of the assumptions underlying the model of communication. To start with, we assume that in the physical layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth. In many cases, the physical and data link layer processes will be running on a processor inside a special network I/O chip and the network layer code will be running on the main CPU. However, other implementations are also possible (e.g., three processes inside a single I/O chip; or the physical and data link layers as procedures called by the network layer process). In any event, treating the three layers as separate processes makes the discussion conceptually cleaner and also serves to emphasize the independence of the layers.

Another key assumption is that machine A wants to send a long stream of data to machine B, using a reliable, connection-oriented service. Later, we will consider the case where B also wants to send data to A