Lecture (2)

Error Detection and Correction Techniques

Three techniques for detecting errors in the transmitted data

5.2.1 Parity Checks

Perhaps the simplest form of error detection is the use of a single **parity bit**. Suppose that the information to be sent, *D* in Figure 5.4, has *d* bits.

In **an even parity scheme**, the sender simply includes one additional bit and chooses its value such that the total number **of 1's in the *d* + 1 bits** (the original information plus a parity bit**) is even**.

For **odd parity schemes**, the parity bit value is chosen such that there is an odd number of 1s. **Figure 5.4 illustrates an even parity scheme**, with the single parity bit being stored in a separate field**.**

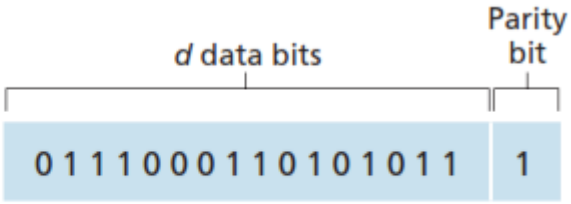


Figure 5.4 ♦ One-bit even parity

The receiver need only count the number of 1s in the received *d* + 1 bits. If an odd number of 1valued bits are found with an even parity scheme, the receiver **knows that at least one bit error** has occurred. More precisely, it knows that **some *odd* number of bit errors has occurred**.

Question

But what happens if an even number of bit errors occurs? For example two bits of value 1 changed to value 0, in this case still the number of 1's is even?

Answer

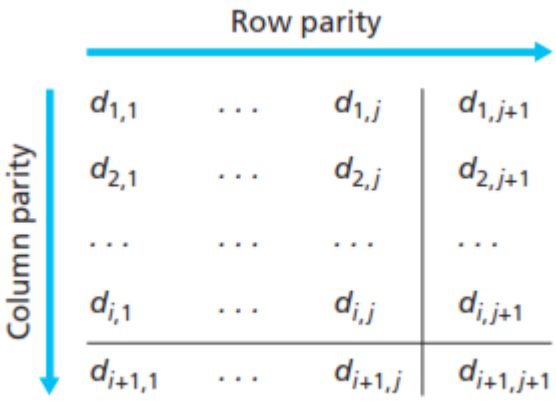
You should convince yourself that this would result in an undetected error.

Clearly, a more robust error-detection scheme is needed.

But before examining error-detection schemes that are used in practice, let's consider a simple generalization of one-bit parity that will provide us with insight into error-correction techniques.

Two-Dimensional Parity Scheme

Figure 5.5 shows **a two-dimensional generalization of the single-bit parity scheme**. Here, the *d* bits in *D* are **divided into *i* rows and *j* columns**. A parity value is computed for each row and for each column. The **resulting *i* + *j* + 1 parity bits** comprise the link-layer frame's error-detection bits.



Example

Suppose now that **a single bit error occurs in the original *d*** bits of information, then **the parity of both the column and the row containing the flipped bit will be in erro**r. The receiver can thus not only ***detect* the fact that a single** bit error has occurred, but can use the column and row indices of the column and row with parity errors to actually identify the bit that was corrupted and *correct* that error!

Figure 5.5 shows an example in which the 1-valued bit in position (2,2) is corrupted and switched to a 0—an error that is both detectable and correctable at the receiver.

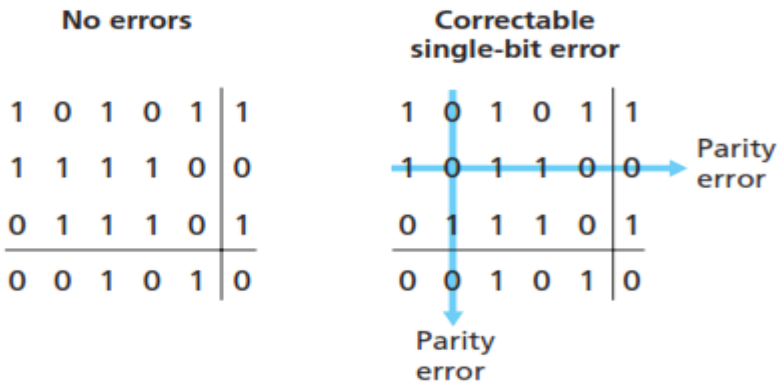


Figure 5.5 ♦ Two-dimensional even parity

The ability of the receiver to both detect and correct errors is known as **Forward Error Correction (FEC)**.

5.2.2 Checksumming

In checksumming techniques, **the *d* bits of data in Figure 5.4 are treated as a sequence of *k*-bit integers**. One simple checksumming method is to simply **sum these *k*-bit integers and use the resulting sum as the error-detection bits**.

The **Internet checksum method** **used by TCP** is based on this approach—

1- Bytes of data are treated as 16-bit integers and summed.
2- The 1s complement of this sum then forms the Internet checksum that is carried in the segment header**.**

The receiver checks the checksum by

1- Taking the 1s complement of the sum of the received data **(including the checksum)**
2- Checking whether the result is all 1 bits.
3- If any of the bits are 0, an error is indicated.

5.2.3 Cyclic Redundancy Check (CRC)

An error-detection technique used widely in today's computer networks is **based on cyclic redundancy check (CRC) codes.**

How CRC codes operate?

1- The sender and receiver must first agree on **an *r* + 1 bit** pattern, known as a **generator**, which we will denote as *G*. We will require that **the most significant (leftmost)** bit of *G* be a 1.

2- Suppose D is the *d*-bit piece of data that the sending node wants to send to the receiving node.

3- **The Sender** will choose *r* additional bits, *R*, and append them to *D* such that the resulting *d* + *r* bit pattern (interpreted as a binary number) is exactly divisible by *G* (i.e., has no remainder) **using modulo-2 arithmetic**.

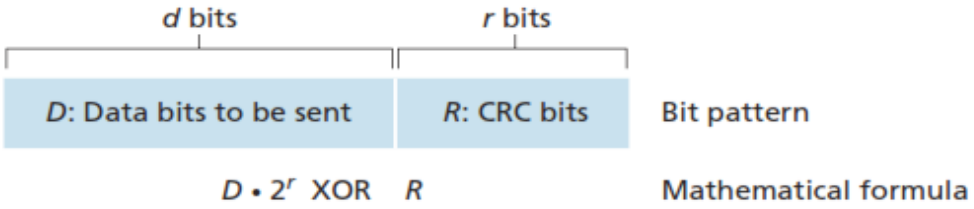


Figure 5.6 ♦ CRC

4- **The Receiver** divides the *d* + *r* received bits by *G*.

5- **If** the remainder is nonzero, the receiver knows that an error has occurred; otherwise the data is accepted as being correct.

What is modulo-2 arithmetic?

All CRC calculations are done in modulo-2 arithmetic without carry in addition or borrow in subtraction. This means that addition and subtraction are identical, and both are equivalent to the bitwise exclusive-or (XOR) of the operands.

Thus, for example,

1011 + 0101 = 1110
1001 + 1101 = 0100
1011 - 0101 = 1110
1001 - 1101 = 0100
1011 XOR 0101 = 1110
1001 XOR 1101 = 0100

Multiplication and division are the same as in base-2 arithmetic, except that any required addition or subtraction is done without carries or borrows.

How the Sender Calculate the R with r bits?

Remember that the receiver treats the pattern d+r pattern bits as a binary number.

The Sender has to calculate R with length r-bits such that the resulting number - after appending r to d- with d+r bits is divisible by G.

In binary Arithmetic it is known that

Any binary number D with length d when multiplied by 2^r will result in a binary string with length d+r.

Thus the quantity $D * 2^r$ XOR R is a number with length d+r.

The sender want that this new number divides G without remainder, that is

$D * 2^r$ XOR R = nG , n=1,2,3 ….

Now if we add R to both sides we get

$D * 2^r$ XOR R + R = nG + R

But addition module 2 is the same as XOR, thus

$D * 2^{-r}$ - nG = nG + R - nG
There fore

$R = D * 2^r$ / G

Figure 5.7 illustrates this calculation for the case of $D = 101110$, $d = 6$, $G = 1001$, and $r = 3$. The 9 bits transmitted in this case are 101110 011. You should check these calculations for yourself and also check that indeed $D *2^r = = 101011 * G$ XOR R

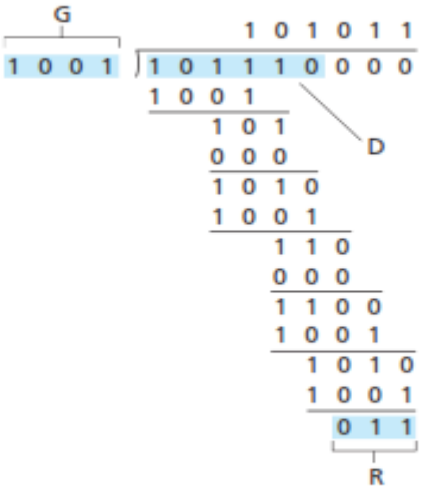


Figure 5.7 ♦ A sample CRC calculation

The CRC is widely used in practice (Ethernet, 802.11 WiFi)