University of Babylon
College of Information Technology
Department of Information Networks
Data Communication and Networking I
4th Stage, Lecture 2

Error-Detection and -Correction Techniques

1. Introduction

In this lecture we will examine a few of the simplest techniques that can be used to detect and, in some cases, correct such bit errors. Figure 1 illustrates the setting for our study. At the sending node, data, *D*, to be protected against bit errors is augmented with error-detection and correction bits (*EDC*). Typically, the data to be protected includes not only the datagram passed down from the network layer for transmission across the link, but also link-level addressing information, sequence numbers, and other fields in the link frame header. Both *D* and *EDC* are sent to the receiving node in a link-level frame. At the receiving node, a sequence of bits, *D′* and *EDC′* is received. Note that *D′* and *EDC′* may differ from the original *D* and *EDC* as a result of in-transit bit flips. The receiver's challenge is to determine whether or not *D′* is the same as the original *D*, given that it has only received *D′* and *EDC′*. The exact wording of the receiver's decision in Figure 1 (we ask whether an error is detected, not whether an error has occurred!) is important. Error-detection and -correction techniques allow the receiver to sometimes, *but not always*, detect that bit errors have occurred. Even with the use of error-detection bits there still may be **undetected bit errors**; that is, the receiver may be unaware that the received information contains bit errors.

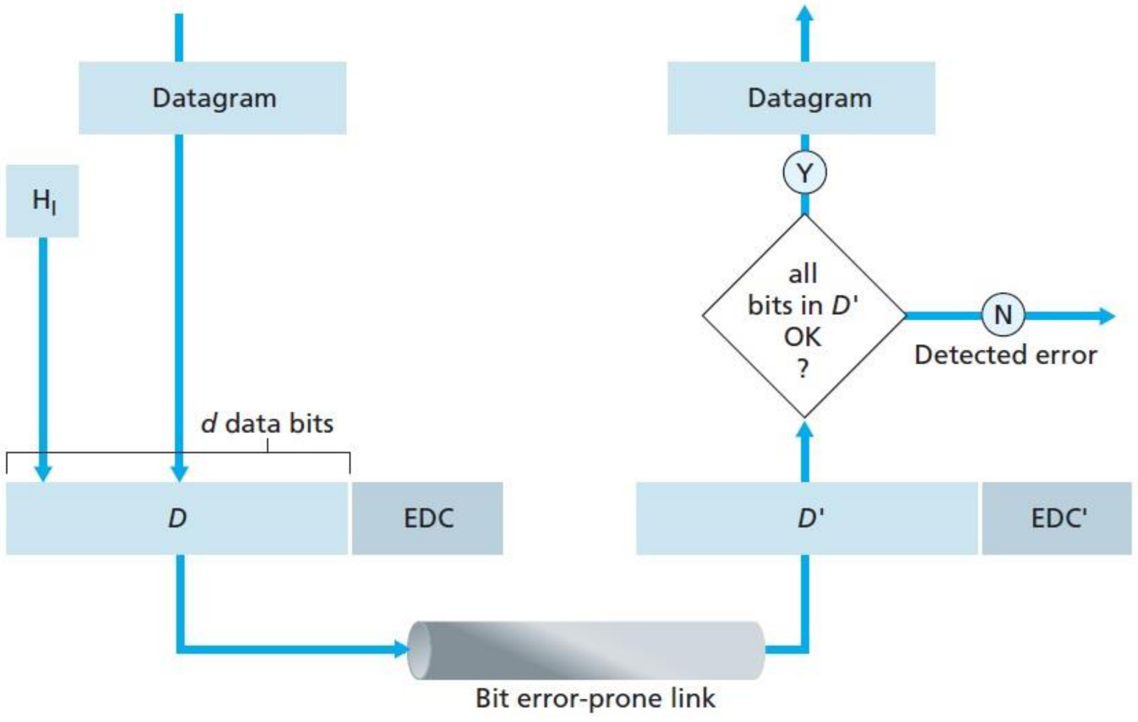


Figure 1: Error-detection and -correction scenario.

Let's now examine three techniques for detecting errors in the transmitted data: parity checks (to illustrate the basic ideas behind error detection and correction), checksumming methods (which are more typically used in the transport layer), and cyclic redundancy checks (which are more typically used in the link layer in an adapter).

1.1 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 2, 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 1s 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 2 illustrates an even parity scheme, with the single parity bit being stored in a separate field.

Receiver operation is also simple with a single parity bit. The receiver need only count the number of 1s in the received *d* + 1 bits. If an odd number of 1-valued 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 have occurred. But what happens if an even number of bit errors occur? You should convince yourself that this would result in an undetected error.

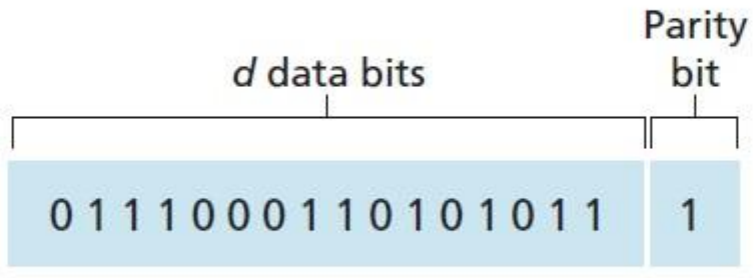
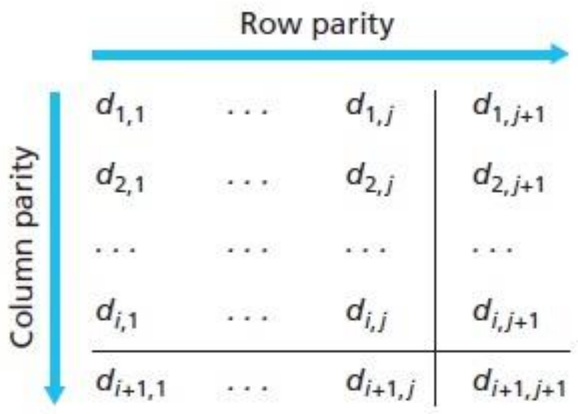


Figure 2: One-bit even parity.

Let's consider a simple generalization of one-bit parity that will provide us with insight into error-correction techniques. Figure 3 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.

Suppose now that a single bit error occurs in the original *d* bits of information. With this **two-dimensional parity** scheme, the parity of both the column and the row containing the flipped bit will be in error. 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 3 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. Although our discussion has focused on the original *d* bits of information, a single error in the parity bits themselves is also detectable and correctable.



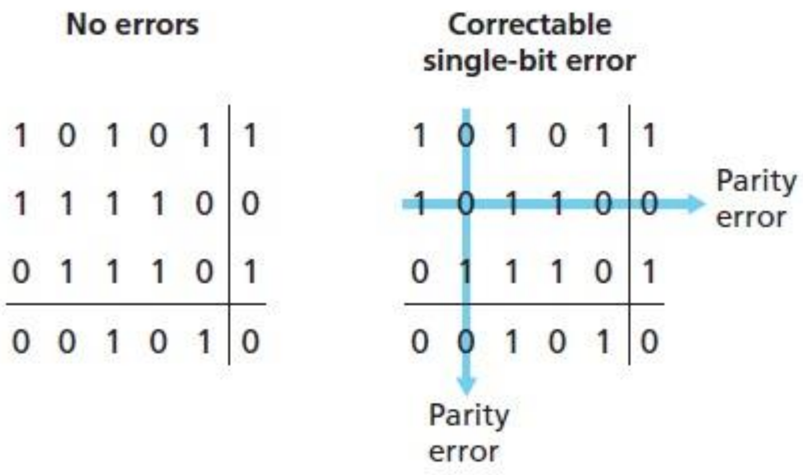


Figure 3: Two-dimensional even parity.

1.2 Checksumming Methods

In checksum error detection scheme, the data is divided into k segments each of m bits. In the sender’s end the segments are added using 1’s complement arithmetic to get the sum. The sum is complemented to get the checksum. The checksum segment is sent along with the data segments as shown in the following example. At the receiver’s end, all received segments are added using 1’s complement arithmetic to get the sum. If the result is ones, the received data is accepted; otherwise discarded.

Suppose that you have the following data: (10110011101010110101101011010101) and k=4, m=8. To calculate the checksum (in the sender side) we do the following calculation:

Sender:

```
     10110011
     10101011
     --------
     01011110
            1
     --------
     01011111
     01011010
     --------
     10111001
     11010101
     --------
     10001110
            1
     --------
 Sum:10001111
Checksum:01110000
```

Receiver:

```
     10110011
     10101011
     --------
     01011110
            1
     --------
     01011111
     01011010
     --------
     10111001
     11010101
     --------
     10001110
            1
     --------
     10001111
     01110000
     --------
 Sum:11111111
   No error
```

1.3 Cyclic Redundancy Check (CRC)

An error-detection technique used widely in today's computer networks is based on **cyclic redundancy check (CRC) codes**. CRC codes operate as follows. Consider the *d*-bit piece of data, *D*, that the sending node wants to send to the receiving node. 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. The key idea behind CRC is that for a given piece of data, *D*, 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. The process of error checking with CRCs is thus simple: The receiver divides the *d* + *r* received bits by *G*. If the remainder is nonzero, the receiver knows that an error has occurred; otherwise the data is accepted as being correct.
Figure 4 illustrates this calculation for the case of *D* = 101110, *d* = 6, *G* = 1001, and *r* = 3. The 9 bits transmitted in this case are 101 110 011.

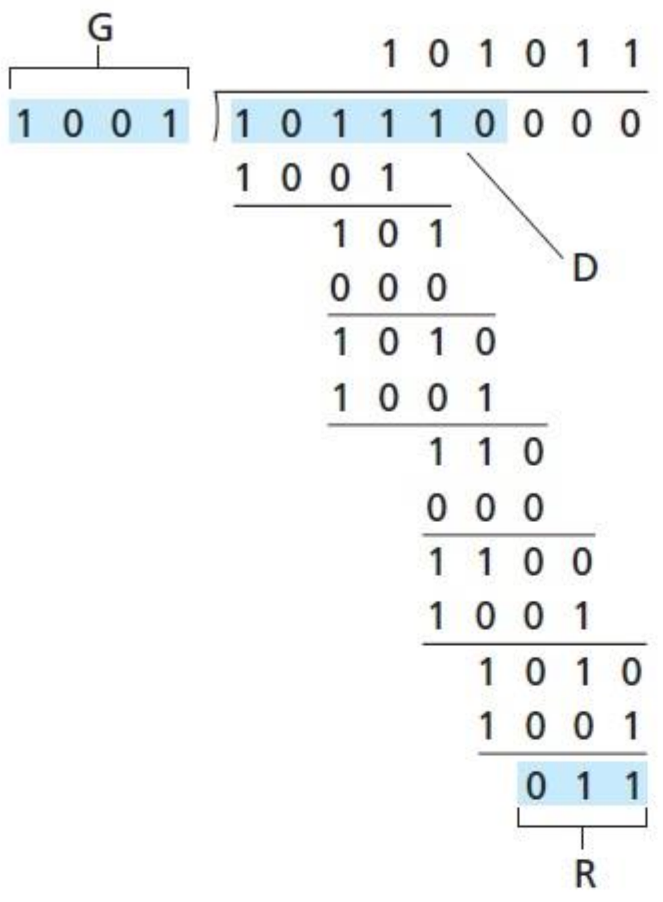


Figure 4: A sample CRC calculation