ERROR DETECTION & CONTROL

Datacan be corrupted during transmission. Some Applications are required that can detect errors and correct them.

INTRODUCTION

- When bits flow from one point to another, they are subjected to unpredictable changes because of interference. This interference can change the shape of signals.

SINGLE-BIT ERROR

- Only 1 bit of a given data unit(byte, character, packet) is changed from 1 -> 0 or 0 -> 1.
- These are the least likely type of errors.
- Example - data sent at the speed of 1 microsecond. For single-bit error to occur, duration of noise must be 1 microsecond, a rare case, normally noises last much longer than this.

BURST ERROR

- Means that 2 or more than 2 bits are changed in the data unit.
- It is not necessary that errors need to occur in consecutive bits.
- Length of burst = last corrupted bit place - first corrupted bit place.
- More likely to occur than the single-bit error. The duration of noise is normally longer than the duration of 1 bit, which means it affects a set of bits.
- No. of bits affected depends on the data rate and the duration of noise.
- Example - 1. data rate = 1 kbps = 1000 bps ; noise = 11100 microseconds ; bits affected = 1000 * 0.00111 = 10 bits.

REDUNDANCY

- To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

DETECTION VERSUS CORRECTION

The difference between **error detection** and **error correction** lies in their complexity and purpose:

1. **Error Detection:**
    - The goal is to check if any errors have occurred in the data.
    - It provides a simple "yes" or "no" answer.

    - The number or type of errors (single-bit or burst errors) doesn't matter; only their presence is checked.
2. **Error Correction:**
    - The goal is to identify and fix errors in the data.
    - This requires knowing:
        - The exact number of errors.
        - The exact locations of the errors in the data.

    - The complexity increases exponentially with the number of errors and the size of the data:
        - For 1 error in an 8-bit unit, there are 8 possible locations.
        - For 2 errors in the same unit, there are 2^8 = 28 possibilities.
        - Correcting 10 errors in a 1000-bit data unit involves an overwhelming number of possibilities, making the process very complex for the receiver.

Thus, while detection is simpler and focuses on identifying errors, correction is much more challenging due to the need for precise error localization and fixing.

FORWARD ERROR CORRECTION VERSUS RETRANSMISSION

Two main methods of error correction-

1. **Forward error correction** - the method where the receiver tries to guess the message by using redundant bits. This is possible only when the number of errors is small.
2. **Retransmission** - the receiver detects the presence of error and asks the sender to resend the message. Done until the receiver gets an error-free message.

CODING

Redundancy is a key concept in error detection and correction, achieved through **coding schemes**. Here's how it works:

1. **Adding Redundant Bits:**
    - The sender adds extra bits, called **redundant bits**, to the actual data bits.
    - These bits create a specific relationship between the data and the redundancy.

2. **Error Checking:**
    - The receiver examines the relationship between the redundant bits and the data bits.

- Any mismatch indicates the presence of errors, allowing detection or correction.

3. **Factors in Coding Schemes:**
   - **Ratio of Redundant Bits to Data Bits:** A higher ratio generally means better error-handling but increases data size.
   - **Robustness of the Process:** Determines how effectively errors are detected or corrected.

There are two types of coding - block coding and convolution coding.

BLOCK CODING

1. Message is divided into blocks of **k bits** each, they are called datawords.
2. **r** redundant bits + **k** datawords bits = **n bits** - codewords.
3. Possible combinations of datwords = 2^k ;
   Possible combinations of codewords = 2^n ;
   N > k ; in block coding , the same dataword is encoded as the same codeword.
   The remaining 2^n - 2^k codewords that are not used are called invalid/illegal.

ERROR DETECTION
When the given two conditions are met, the receiver detects a change in the original codeword.

1. The receiver has(or can find)a list of valid codewords.
2. The Original Codeword Has Changed To An Invalid One.

Process -
1. Sender creates codewords from datawords by using a generator that applies the rules and procedures of encoding.
2. Each codeword sent to the receiver may change during transmission.
3. If received codeword = valid codeword, word is accepted, and corresponding dataword is extracted for the use. If not valid, the codeword is discarded.
4. If the codeword is corrupted, but the received word still matches a valid codeword, the error remains undetected.
5. Block coding can only detect single errors.

ERROR CORRECTION

1. Comparing the received codeword with the first code word in the table (01001 versus 00000), the receiver decides that the first code word is not the one that was sent because there are two different bits.
2. By The Same Reasoning, the original codeword cannot be the third or fourth one in the table.

3. The original codeword must be the second one in the table because this is the only one that differs from thereceivedcodewordby1 bit.The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.

HAMMING DISTANCE

- Hamming dist. = number of differences between corresponding bits.
- between the two words(of same size).
- Shown by d(x,y).
- XOR operation is used to determine the hamming distance.
- Example - d(000, 011) = 2 {000 XOR 011 = 011}

MINIMUM HAMMING DISTANCE

- The smallest hamming distance between all the possible pairs.

HAMMING DISTANCE AND THE ERROR

- When the codeword is corrupted, hamming dist. = number of bits affected by the error.
- Hamming dist. = number of bits that are corrupted during the transmission.

LINEAR BLOCK CODES

- Code in which the exclusive OR(addition modulo2) - XOR of two valid codewords creates another valid codeword.

SIMPLE PARITY-CHECK CODE

- Most familiar error-detecting code.
- Parity bit = codeword - dataword - selected in a way that number of 1s in codeword = even/odd.
- A simple parity-check code is a single-bit error-detecting code in which n =k + 1 with dmin =2.

How Parity Works:

1. **At the Sender:**
    - The sender calculates the **parity bit** by adding the 4 data bits using **modulo-2 addition** (XOR operation).
    - If the total number of 1s in the data bits is:
        - **Even**: The parity bit is **0** (no need to add another 1).
        - **Odd**: The parity bit is **1** (to make the total even).
    - The codeword sent consists of the 4 data bits and the parity bit. This ensures that the total number of 1s in the codeword is always even.

2. **At the Receiver:**
    - The receiver receives a 5-bit codeword (4 data bits + 1 parity bit). It might be corrupted during transmission.
    - A **syndrome** is calculated by performing the same modulo-2 addition (XOR) over all 5 bits of the received codeword:
        - **Syndrome = 0**: The total number of 1s in the received codeword is even, indicating no error.
        - **Syndrome = 1**: The total number of 1s is odd, indicating an error.

3. **Decision Logic:**
    - If the **syndrome** is **0**, the codeword is accepted, and the 4 data bits are extracted as the valid dataword.
    - If the **syndrome** is **1**, the codeword is discarded, and the dataword is not created, as an error has occurred.

Parity checking is a simple way to detect **single-bit errors** during transmission. However, it cannot detect errors if multiple bits are corrupted in a way that keeps the total number of 1s even.

HAMMING CODE

Hamming codes are a class of **error-correcting codes** that can detect and correct errors in transmitted data. The most common Hamming codes are designed with a minimum Hamming distance (d_{min}) of 3, meaning they can:

- **Detect up to 2-bit errors** (since $d_{min} >= s + 1$, where s is the number of errors detected).

- **Correct 1-bit errors** (since $d_{min} >= 2t + 1$, where t is the number of errors corrected).

Structure of Hamming Codes

1. **Parameters**:

   - Choose an integer m>=3 , where m is the number of **parity bits**.

   - The codeword length n = 2^m - 1.

   - The number of data bits k = n - m.

   - The number of parity bits r = m.

2. **Parity-Check Equations:**

   - The parity bits are computed using subsets of the data bits.

   - For the (7,4) code:

     - r_0 = a_2 + a_1 + a_0 \mod 2,

     - r_1 = a_3 + a_2 + a_1 \mod 2,

     - r_2 = a_1 + a_0 + a_3 \mod 2.

3. **Codeword Construction:**

   - The codeword is formed as [a_3, a_2, a_1, a_0, r_2, r_1, r_0].

Error Detection and Correction

1. Syndrome Calculation:

   - The decoder computes a **syndrome** (s_2, s_1, s_0) using the received bits:

     - s_0 = b_2 + b_1 + b_0 + q_0 \mod 2 ,

     - s_1 = b_3 + b_2 + b_1 + q_1 \mod 2 ,

- s_2 = b_1 + b_0 + b_3 + q_2 \mod 2.

2. **Syndrome Interpretation:**

- The syndrome helps identify the position of the error (if any):

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	ql	b_2	q2	b_O	b_3	b_1

3. **Error Correction:**

- If the syndrome is non-zero, the corresponding bit is flipped to correct the error.

Example: Hamming (7,4) Code

- Dataword (4 bits): 1011 .

- Codeword (7 bits): Compute parity bits:

- r_0 = a_2 + a_1 + a_0 = 0 + 1 + 1 = 0 ,

- r_1 = a_3 + a_2 + a_1 = 1 + 0 + 1 = 0 ,

- r_2 = a_1 + a_0 + a_3 = 1 + 1 + 1 = 1 .

- Final Codeword: [1, 0, 1, 1, 1, 0, 0].

Limitations

1. **Double Errors**:

- If **two bits are flipped**, the syndrome may incorrectly correct a wrong bit.

- Hence, Hamming codes are only **guaranteed to correct single-bit errors**.

2. **Error Detection vs. Correction**: For error detection only, a different design (e.g., extended Hamming code) is needed to detect more errors.

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) is a **powerful error-detection technique** widely used in digital networks (LANs, WANs) and storage systems. Unlike simple parity checks, CRC can detect **burst errors** and is highly efficient in hardware implementations.

1. Basic Idea of CRC

- **Goal**: Detect errors in transmitted data by appending a **checksum (CRC code)** to the original data.

- **Method**: Treat data as a binary polynomial and divide it by a predefined **generator polynomial** {g(x)}. The remainder becomes the CRC.

Example: CRC Code (7,4)

- Dataword (k=4 bits): e.g. - `1001`

- Codeword (n=7 bits): Augment dataword with `n-k=3` zeros → `1001000`

- Generator polynomial (divisor): e.g. - `1011` (binary) = x^3 + x + 1

- **Compute remainder** - when dividing augmented dataword by generator → CRC bits.

- Final codeword: Dataword + CRC bits (e.g., `1001` + `101` → `1001101`).

2. CRC Encoder & Decoder

Encoder Steps:

1. Augment dataword - with `n-k` zeros.

2. Divide - by generator polynomial (modulo-2 division).

3. Append remainder (CRC) - to dataword.

Decoder Steps:

1. Divide received codeword by generator polynomial.

2. If remainder = 0 → No error detected.

3. If remainder ≠ 0 → Error detected (retransmit or discard).

3. Polynomial Representation

- Binary data can be represented as polynomials:

  - `1011` → x^3 + x + 1

- **Modulo-2 arithmetic (XOR operations)**:

- Addition & subtraction are the same (XOR).
- No carry/borrow in calculations.

Example of Polynomial Division

- Dividend (Augmented dataword): (x^6 + x^3) (`1001000`)
- Divisor (Generator): (x^3 + x + 1) (`1011`)
- Perform division:

  1001000 ÷ 1011 = Quotient (discarded) + Remainder (CRC)

- **Final CRC**: Remainder (`101`).

4. Error-Detection Capabilities

CRC can detect:

1. All single-bit errors (if generator has ≥ 2 terms).
2. All double-bit errors (if generator does not divide \(x^t + 1 \) for \(t \leq n-1 \)).
3. Any odd number of errors (if generator includes \(x+1 \) as a factor).
4. Burst errors of length ≤ `r` (where `r` is CRC length).

Example: Detecting Burst Errors

- Generator: `1011` (3-bit CRC)
- Burst error of length ≤ 3: Always detected.
- Burst error of length 4: Detected with probability \(1 - 2^{-(r-1)} \).

5. Standard CRC Polynomials

Commonly used CRC polynomials in networking:

CRC-8	x^8 + x^2 + x + 1 \)	ATM headers
CRC-16	x^{16} + x^{12} + x^5 + 1	USB, Bluetooth

| CRC-32 | x^{32} + x^{26} + \dots + 1 | Ethernet, ZIP, PNG |

6. Advantages of CRC

✅ High error-detection capability (catches burst errors).

✅ Efficient hardware implementation (shift registers & XOR gates).

✅ Low computational overhead (fast in software).

7. Checksum (Alternative to CRC)

- Checksum is a simpler error-detection method (used in TCP/IP).

- **Steps**:

  1. Divide data into 16-bit words.
  2. Sum all words (one's complement arithmetic).
  3. Complement the sum → Checksum.

- **Receiver verifies checksum** (if sum + checksum = 0, no error).

Limitations of Checksum:

❌ Weaker than CRC (misses some errors like swapped bits).

❌ Replaced by CRC in modern protocols (e.g., Ethernet uses CRC-32).

Conclusion

- **CRC** is a **robust** error-detection method for networks and storage.

- **Polynomial division** ensures high reliability.

- **Checksum** is simpler but less reliable (used in higher layers like TCP).