

Lecture 5 – Link Layer

Error control coding:

Goal: decrease the residual bit error rate (BER) without reducing throughput too much

Key idea: add redundant bits to allow detection and/ or correction of errors (basically this is a code word
--> the combination of the actual data and the error bit)

Fundamental tradeoff:

- Increasing redundant bits improves BER
- Decreases throughput

In the stack:

Application → transport → network → link:error control → physical

Block codes: (n,k) notation: encode words so that you can transmit them and also add error detection

Error control coding framework:

- Take k data bits (data) → transmit an n bit codeword
- There are 2^k valid codewords out of 2^n possible n bit words
- The remaining $2^n - 2^k$ words are non codewords (invalid)
- This is called an (n,k) block code

Encoding: calculate word = $g(\text{data})$ and transmit word

Common structure: word = data | f(data)

Code rate:

Rate of the code: $R_c = k/n = \text{data bits/ code word}$

- $R_c = 1$: no redundancy (no error protection) Redundancy in code rate refers to the extra bits added to data for error detection and correction
- $R_c < 1$: redundancy added (throughput reduced by factor R_c)
- R_c lower: more redundancy → potentially better error protection

Decoding: at receiver, calculate data= g^{-1} (received word)

Error control options:

- Error correction (FEC) → Send extra information so the receiver can fix small mistakes on its own.
 - Any received word is mapped to the nearest valid codeword
 - No retransmission needed
 - Higher overhead (more redundant bits)
- Error detection (ARQ) → Don't try to fix errors – just detect them and ask for the message again.
 - Only the 2^k valid codewords are accepted
 - Invalid words trigger a retransmission request
 - Lower overhead but requires a feedback channel
- Combined detection and correction:

If the mistake is small → fix it.

If it's big → ask for a resend.

- Some non code words are corrected
 - Some non code words are rejected
 - All valid codewords are accepted

Here $n - k = 1$: send k data bits followed by a parity bit forcing an **even number of ones**

Data (3 bits)	Parity Bit	Codeword (4 bits)
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Example: parity bit code:

Properties: Rate $R_c = k/(k + 1)$ (efficient). Detects any odd number of errors. Cannot correct errors.

Here $k = 1$ and n can be chosen as needed

Two codewords: n zeros or n ones

For $n = 3$ repetition code:

- Codewords: $\{000, 111\}$
 - Rate: $R_c = 1/3$ (very inefficient)
 - $d_{\min} = 3$

Options:

- Can detect up to 2 errors ($e_c = 0, e_d = 2$)
 - **OR** correct 1 error ($e_c = 1, e_d = 1$)

Example: repetition code:

General repetition by n : Rate = $1/n$, $d_{\min} = n$

Hamming distance: Hamming Distance refers to the number of positions at which two strings of the same length differ.

hamming distance between two words is the number of bit positions in which they differ

Example: $\text{distance}(0011, 1010) = 2 \rightarrow$ since they differ in the 1st and 4th positions

Interpretation: think of codewords as points in a n-dimensional binary space → hamming distance is the city block distance in this space

Minimum distance of a code: a **code** is a set of n bit codewords \rightarrow code is set of n bit actual data + redundancy data

Minimum distance d_{\min} is the smallest hamming distance between any pair of distinct codewords

$$d_{\min} = \min_{C_i \neq C_j \in \mathcal{C}} \text{dist}(C_i, C_j)$$

Why?

- It completely determines the error detection and correct capability
 - Larger d_{\min} → more powerful code
 - Typically requires more redundancy

Consider the code $\mathcal{C} = \{000, 011, 101, 110\}$:

Pair	Distance
dist(000, 011)	2
dist(000, 101)	2
dist(000, 110)	2
dist(011, 101)	2
dist(011, 110)	2
dist(101, 110)	2

Therefore $d_{\min} = 2$.

Computing d_{\min} example: **Note:** $\binom{4}{2} = 6$ pairs to check for 4 codewords. In general, $\binom{|\mathcal{C}|}{2}$ pairs.

Code power: e_c and e_d

$e_c \rightarrow$ the number of errors guaranteed to be corrected successfully

$e_d \rightarrow$ number of errors guaranteed to be detected successfully

Requirements:

- $e_c < e_d \rightarrow$ if we can correct we must first detect
- The pair $(e_c, e_d) \rightarrow$ describes the receiver's error control strategy

Given a code with known d_{\min} what (e_c, e_d) pairs are valid?

Receiver algorithm:

Let e be the apparent number of errors in received word R : $e = \min \text{dist}(R, C)$

Decision rule:

- If $0 \leq e \leq e_c \rightarrow$ correct - pick C_i as the intended codeword
- If $e_c < e \rightarrow$ detect - request retransmission, discard, or flag as uncorrectable

The algorithm is fully described by:

- The code $C \rightarrow$ set of valid codewords
- The parameters e_c and e_d

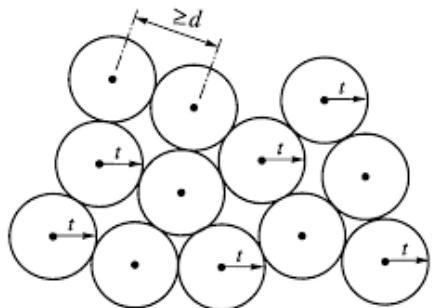
Hamming sphere interpretation:

Visualize each codeword with a 'sphere' of radius e_c :

- all words within distance e_c of codeword C_i corrected to C_i
- Words outside all sphere but within distance e_d of the nearest codeword are detected as errors
- Words farther than e_d from all codewords: undetectable errors

Configurations:

- Combined correction + detection: small spheres, gap between them
- Maximum correction: spheres as large as possible (touching)
- Maximum detection: no spheres ($e_c = 0$), maximum gap



The d_{\min} theorem: for a code with minimum distance d_{\min} , the receiver uses an (e_c, e_d) pair that must satisfy:

$$e_c + e_d \leq d_{\min} - 1$$

$$e_{\epsilon} \leq e_d$$

- Corrected error and detected error has to be less than minimum hamming distance -1
- Corrected error is less than detected error

Why $e_c < e_d$?

If we can correct an error, we must necessarily detect its presence first. Hence $e_c \leq e_d$.

Why $e_c + e_d \leq d_{\min} - 1$?

Suppose $e_C + e_d = d_{\min}$. Consider the two closest codewords C_i and C_j with $\text{dist}(C_i, C_j) = d_{\min}$.

There exists a received word R that is:

- Distance e_c from C_i
 - Distance e_d from C_i

Contradiction:

- If C_i was sent \Rightarrow we should **correct** R to C_i
 - If C_j was sent \Rightarrow we should **detect** the error and reject R
 - Algorithm must choose one action \Rightarrow cannot guarantee **both**

Therefore $e_1 + e_2 \leq d_{\text{min}}$, i.e., $e_1 + e_2 \leq d_{\text{min}} - 1$

Consequences of the d₊₊ theorem

1. Maximum error detection (set $e_c = 0$)
 - a. $(e_d)_{\max} = d_{\min} - 1$
 2. Maximum error correction (set $e_c = e_d$)
 - a. $2e_c \leq d_{\min} - 1 \rightarrow (e_d)_{\max} = \text{floor}((d_{\min} - 1) / 2)$
 3. You cannot simultaneously maximize both
 - a. Must choose between best detection and best correction

Quick reference for choosing d_{\min}

To **detect** d errors: $d_{\min} = d + 1$

To **correct** d errors: $d_{\min} = 2d + 1$

Example 1

Problem: Design a code from 3-bit codewords with 2 codewords.

Questions:

- ① What is the rate of the code?
- ② Which codewords should you send?
- ③ What options exist for error detecting or correcting?

Rate: $R_c = k/n = 1/3$

Codewords: Pick any two "opposites": {000, 111} are convenient. Then $d_{\min} = 3$.

Two sensible options:

Option	(e_c, e_d)	Decoding Rule
Error correction	(1, 1)	{000, 100, 010, 001} → 0 {111, 110, 101, 011} → 1
Error detection	(0, 2)	000 → 0; 111 → 1 All others → retransmit

Example 2

Problem: Design a code using 3-bit codewords with 4 codewords.

Questions:

- ① What is the rate of the code?
- ② Which codewords should you send?
- ③ What options exist for error detecting or correcting?

Rate: $R_c = k/n = 2/3$

Codewords: {000, 011, 101, 110}. Then $d_{\min} = 2$.

Only one sensible option: Error detection with $(e_c = 0, e_d = 1)$

Codeword	Data
000	00
011	01
101	10
110	11
Others	Retransmit

With $d_{\min} = 2$: can detect 1 error, cannot correct any errors.

When to use error correction (FEC):

Error detection with ARQ is usually 3-5x more efficient for typical networking

Use FEC when retransmission is impossible or impractical

Scenario	Reason
Storage (CDs, hard drives, tape)	Errors are permanent
Real-time (voice, video)	No time for retransmission
Simplex (digital broadcast TV)	No return path
High-delay (satellite, deep space)	RTT makes retransmission too slow