# Communication and Error Processing

Communication is central to computing.

ISO: 7 levels of communication protocol.

Here we discuss basics of low level.

Communication passes through a noisy **channel**, which can introduce errors and erasures.

Transmitter sends data X.

Data X passes through channel, which changes or erases some parts of X, resulting in a received data Y.

Errors also occur in memory and disks.

We begin with some error detection and correction basics.

# How to detect and correct errors?

If what we receive is **impossible** we know there is an error.

For example - if we **know** the sender sends only correct English words, anything else is an error.

Fixing an error - find closest
legal replacement.

In general, any bit pattern is possible.

Introducing **redundancy** allows detection and possibly correction.

**Trivial example**: send everything twice.

Anything where we do not get the same thing twice must be an error!

What to do if there is an error?

Simple solution: ask for re-transmission.

# Error correction

Alternately, allow correction at receiver:

Send everything 3 times...

Decide by majority (which is closest match).

Erasure: easier to fix than an error.

What if there is more than 1 error?

Can use even more redundancy...

Problem: very inefficient!

# Error detecting and correcting codes

Much better efficiency when using error detecting codes.

Simplest efficient error correcting code: **parity**

Compute p, parity of bits of X (e.g. p = number of bits of 1 in X, mod 2).

Send code word
C = X concatenated with p

Receive Y - which is assumed be same as C except, possibly, error in 1 bit.

Error detected by finding parity of Y:

Even parity - no error.
Othewise - error.

**Hamming distance** between two words d(a, b) = number of 1 bits in XOR a, b

Distance of code - minimum distance between all pairs of code words.

For parity code, d=2. Any code with d=2 can detect 1 bit error.

With parity, can **fix** one bit **erasure**.

Example: X=10110, parity 1

Code word C=101101 (last bit is parity).

Receive Y=1U1101 (second bit erased),

Erased U bit = parity of other bits.

To **correct** an **error**, need at least d=3.

# Hamming Code

Simple code with d=3

With $r$ parity bits, word size $n = 2^r - r - 1$.

For $r = 3$, we have $n = 4$ data bits:

$$p_0 = X_3 \oplus X_1 \oplus X_0$$
$$p_1 = X_3 \oplus X_2 \oplus X_0$$
$$p_2 = X_3 \oplus X_2 \oplus X_1$$

In receiver, recompute parities from Y.
Compare to actual values of $p_2, p_1, p_0$ in Y.
XOR of bit vectors is called **syndrome**.

Simple algorithm - order bits:

$$X_3 \quad X_2 \quad X_1 \quad p_2 \quad X_0 \quad p_1 \quad p_0$$

Non-zero syndrome is position of error bit (rightmost is position 1).

# Beyond Hamming Code

Can add 8th bit, overall parity: d=4

Will correct 1 error, and detect 2.
(or fix 3 erasures)

k-dimensional-array codes: d=k+1

| | | | |
|---|---|---|---|
| | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |

Codes with any required $d$ available.

Efficiency improves with code size.

Encoding, and esp. decoding complexity may be high.

# Parallel Communications

Used for fast data transfer - short distance.

Wasteful in hardware - esp. wires.

Requires **handshake** to synchronize.

Control signals to latch data, ready signal.

# Serial Communications - USART

Use of bit-wide com. saves wires.

Conversion: parallel to serial, and vice versa.

Usually done by special hardware.

USART allows several communication rates.

Can also encode/decode with simple codes.

Other possibilities: MODEM, ETHERNET.

## Signal level: RS232, RS422

RS232: common (early) terminal standard.

Signal is +/-12V

Asynchronous: **start bit**,
1 or more **stop bits**.

Other standards: current loop, differential.

## Hardware/Software Handshakes

Optional hardware hand-shake: DSR, CTS

Software handshake: "X-on/X-off".

Usually done by sending "CTRL S" character for X-off, and "CTRL Q" character for X-on.