

INTRODUCTION TO EECS II DIGITAL COMMUNICATION SYSTEMS

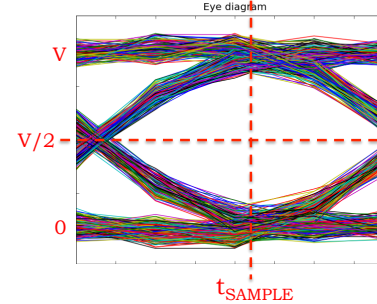
6.02 Fall 2009 Lecture #7

- Dealing with errors through channel coding
- Hamming distance & error detection
- Parity, Checksums, CRC
- Error correction using multiple parity bits
- (n,k) block codes

6.02 Fall 2009

Lecture 7, Slide #1

There's good news and bad news...



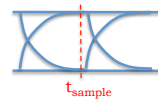
The good news: Our digital signaling scheme usually allows us to recover the original signal despite small amplitude errors introduced by inter-symbol interference and noise. An example of the digital abstraction doing its job!

The bad news: larger amplitude errors (hopefully infrequent) that change the signal irretrievably. These show up as **bit errors** in our digital data stream.

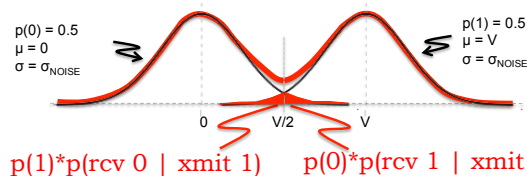
6.02 Fall 2009

Lecture 7, Slide #2

Bit Errors



Assuming a Gaussian PDF for noise and only 1-bit of inter-symbol interference, samples at t_{SAMPLE} have the following PDF:



We can estimate the bit-error rate (BER) using Φ , the unit normal cumulative distribution function:

$$BER = (0.5)\Phi\left[\frac{V/2 - V}{\sigma_{\text{NOISE}}}\right] + (0.5)\left[1 - \Phi\left[\frac{V/2 - 0}{\sigma_{\text{NOISE}}}\right]\right] = \Phi\left[\frac{-V/2}{\sigma_{\text{NOISE}}}\right]$$

For a smaller BER, you need a smaller σ_{NOISE} or a larger V !

6.02 Fall 2009

Lecture 7, Slide #3

BER for common channels

Channel type	Bandwidth	BER
Telephone Landline	2 Mbits/sec	10^{-4} to 10^{-6}
Twisted pair (differential)	1 Gbits/sec	$\leq 10^{-7}$
Coaxial cable	100 Mbits/sec	$\leq 10^{-6}$
Fiber Optics	10 Tbits/sec	$\leq 10^{-9}$
Infrared	2 Mbits/sec	10^{-4} to 10^{-6}
3G cellular	1 Mbits/sec	10^{-4}

Source: Rahmani, et al, *Error Detection Capabilities of Automotive Technologies and Ethernet – A Comparative Study*, 2007 IEEE Intelligent Vehicles Symposium, p 674-679

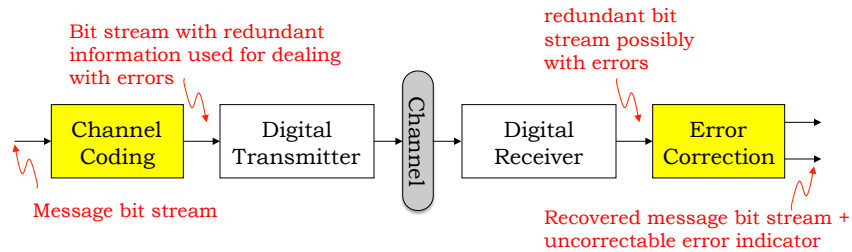
$$p(\text{error in 1kByte packet} \mid \text{BER} = 10^{-6}) = 1 - (.999999)^{8192} = \sim .008$$

6.02 Fall 2009

Lecture 7, Slide #4

Channel coding

Our plan to deal with bit errors:



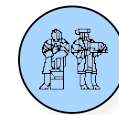
We'll add redundant information to the transmitted bit stream (a process called **channel coding**) so that we can detect errors at the receiver. Ideally we'd like to correct commonly occurring errors, e.g., error bursts of bounded length. Otherwise, we should **detect uncorrectable errors** and use, say, retransmission to deal with the problem.

6.02 Fall 2009

Lecture 7, Slide #5

Error detection and correction

Suppose we wanted to reliably transmit the result of a single coin flip:



Heads: "0"

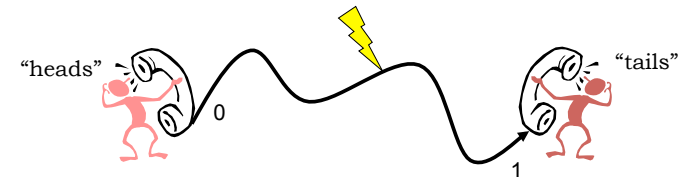


Tails: "1"

This is a prototype of the "bit" coin for the new information economy. Value = 12.5¢

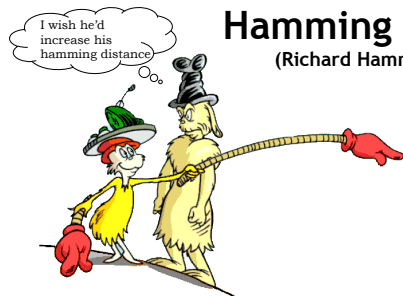


Further suppose that during transmission a **single-bit error** occurs, i.e., a single "0" is turned into a "1" or a "1" is turned into a "0".



6.02 Fall 2009

Lecture 7, Slide #6



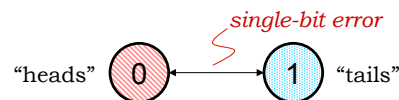
Hamming Distance

(Richard Hamming, 1950)

HAMMING DISTANCE: The number of digit positions in which the corresponding digits of two encodings of the same length are different

The Hamming distance between a valid binary code word and the same code word with single-bit error is 1.

The problem with our simple encoding is that the two valid code words ("0" and "1") also have a Hamming distance of 1. So a single-bit error changes a valid code word into another valid code word...



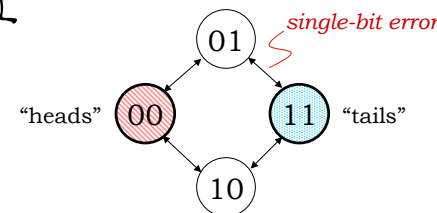
6.02 Fall 2009

Lecture 7, Slide #7

Error Detection



What we need is an encoding where a single-bit error doesn't produce another valid code word.



If D is the minimum Hamming distance between code words, we can detect up to $(D-1)$ -bit errors

We can add single-bit error detection to any length code word by adding a **parity bit** chosen to guarantee the Hamming distance between any two valid code words is at least 2. In the diagram above, we're using "even parity" where the added bit is chosen to make the total number of 1's in the code word even.

6.02 Fall 2009

Lecture 7, Slide #8

Parity check

- A parity bit can be added to any length message and is chosen to make the total number of “1” bits even (aka “even parity”).
- To check for a single-bit error (actually any odd number of errors), count the number of “1”s in the received message and if it’s odd, there’s been an error.

0 1 1 0 0 1 0 1 0 0 1 1 → original word with parity
 0 1 1 0 0 0 0 1 0 0 1 1 → single-bit error (detected)
 0 1 1 0 0 0 1 1 0 0 1 1 → 2-bit error (not detected)

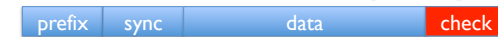
- One can “count” by summing the bits in the word modulo 2 (which is equivalent to XOR’ing the bits together).

6.02 Fall 2009

Lecture 7, Slide #9

Is a Data Packet Error Free?

- Transmitter appends check bytes computed from data to end of packet



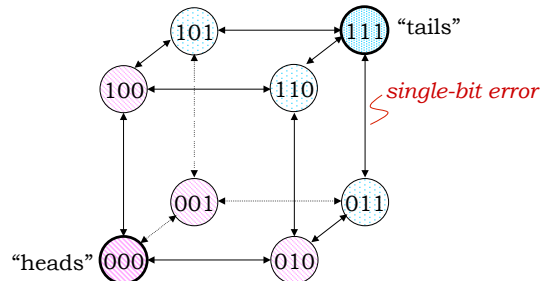
- Receiver recomputes check bytes from received data, compares with received check bytes
 - Mismatch: there’s an error in the received data/check bytes, so discard packet
- Two popular methods for computing check bytes
 - Adler-32
 - Easy in SW, not good for < 500 byte packets
 - Cyclical Redundancy Check (CRC-16, CRC-32)
 - Easy in HW, good for any length packet
 - Good at error bursts

Max packet size (bits)	91	171	268	2974	91.6k	128k
Hamming distance	8	7	6	5	4	3

6.02 Fall 2009

Lecture 7, Slide #10

Error Correction



If D is the minimum Hamming distance between code words, we can correct up to $\left\lfloor \frac{D-1}{2} \right\rfloor$ bit errors

By increasing the Hamming distance between valid code words to 3, we guarantee that the sets of words produced by single-bit errors don’t overlap. So if we detect an error, we can perform **error correction** since we can tell what the valid code was before the error happened.

- Can we safely detect double-bit errors while correcting 1-bit errors?
- Do we always need to triple the number of bits?

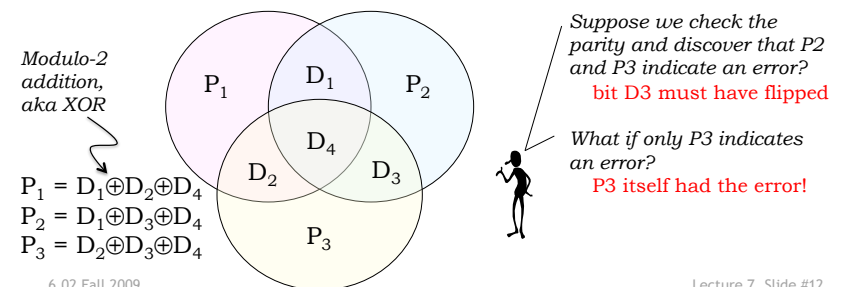
6.02 Fall 2009

Lecture 7, Slide #11

Error Correcting Codes (ECC)

Basic idea:

- Use multiple parity bits, each covering a subset of the data bits.
- No two message bits belong to exactly the same subsets, so a single-bit error will generate a unique set of parity check errors.



6.02 Fall 2009

Lecture 7, Slide #12

Checking the parity

- Transmit: Compute the parity bits and send them along with the message bits
- Receive: After receiving the (possibly corrupted) message, compute a syndrome bit (E_i) for each parity bit. For the code on slide #12:

$$E_1 = D_1 \oplus D_2 \oplus D_4 \oplus P_1$$

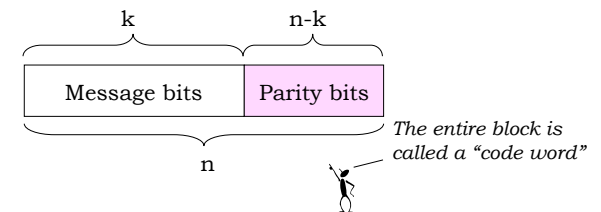
$$E_2 = D_1 \oplus D_3 \oplus D_4 \oplus P_2$$

$$E_3 = D_2 \oplus D_3 \oplus D_4 \oplus P_3$$

- If all the E_i are zero: no errors!
- Otherwise the particular combination of the E_i can be used to figure out which bit to correct.

(n,k) Systematic Block Codes

- Split message into k -bit blocks
- Add $(n-k)$ parity bits to each block, making each block n bits long.



If we want to **correct single-bit errors**, how many parity bits do we need?

How many parity bits to use?

- Suppose we want to do single-bit error correction
 - Need unique combination of syndrome bits for each possible single bit error
 - n -bit blocks $\Rightarrow n$ possible single bit errors
 - Syndrome bits all zero $\Rightarrow$ no errors
- Assume $n-k$ parity bits (out of n total bits)
 - Hence there are $n-k$ syndrome bits
 - $2^{n-k} - 1$ non-zero combinations of p bits
- So, at a minimum, we need $n \leq 2^{n-k} - 1$
 - Given k , use constraint to determine minimum n needed to ensure single-bit error correction is possible
 - (n,k) Hamming SECC codes: (7,4) (15,11) (31,26)

The (7,4) Hamming SECC code is shown on slide 12

Which (n,k) code does one use?

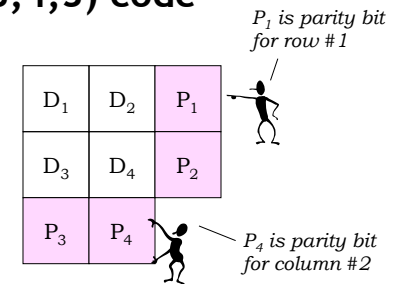
- The minimum Hamming distance d between code words determines how we can use code:
 - To detect D -bit errors: $d > D$
 - To detect and correct C -bit errors: $d > 2C$
 - Sometimes code names include min Hamming distance: (n,k,d)
- With a $(n,k,3)$ code one can
 - Detect 1-bit and 2-bit errors OR
 - Detect and correct 1-bit errors (SEC)
- With a $(n,k,4)$ code one can
 - Detect 1-bit, 2-bit and 3-bit errors OR
 - Detect and correct 1-bit errors, and detect 2-bit errors (SECDED)
- With a (n,k,d) code and for $c < d/2$ one can
 - Detect and correct c -bit errors, and
 - If $c+1 < d/2$, detect $(c+1)$ -bit up to $(d-c-1)$ -bit errors

Error-Correcting Codes

- To efficiently use bandwidth want to maximize a code's *code rate*, defined as k/n . Goal: $k/n \rightarrow 1$
- Parity is a $(n+1, n, 2)$ code
 - Good code rate, but only 1-bit error detection
- Replicating each bit r times is a $(r, 1, r)$ code
 - Simple way to get great error correction; poor code rate
 - Handy for solving quiz problems!
- Hamming single-error correcting codes (SECC) are $(n, n-p, 3)$ where $n = 2^p - 1$ for $p > 1$
 - Adding an overall parity bit makes the code $(n+1, n-p, 4)$

A simple $(8, 4, 3)$ code

Idea: start with rectangular array of data bits, add parity checks for each row and column. Single-bit error in data will show up as parity errors in a particular row and column, pinpointing the bit that has the error.



```
0 1 1
1 1 0
1 0
```

```
0 1 1
1 0 0
1 0
```

```
0 1 1
1 1 1
1 0
```

Parity for each row and column is correct $\Rightarrow$ no errors

Parity check fails for row #2 and column #2 $\Rightarrow$ bit D_4 is incorrect

Parity check only fails for row #2 $\Rightarrow$ bit P_2 is incorrect

If we add an overall parity bit P_5 , we get a $(9, 4, 4)$ code!