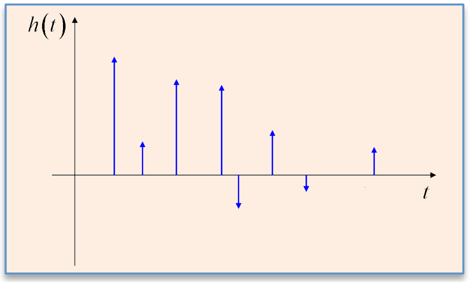
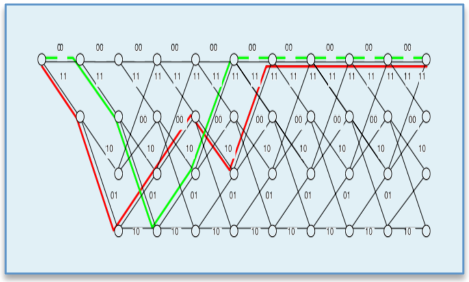
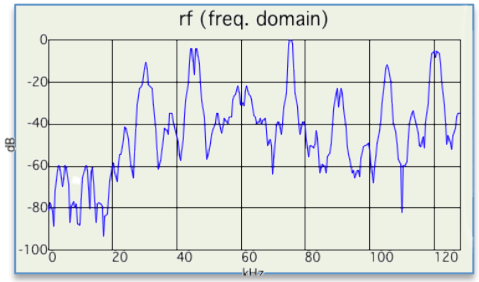
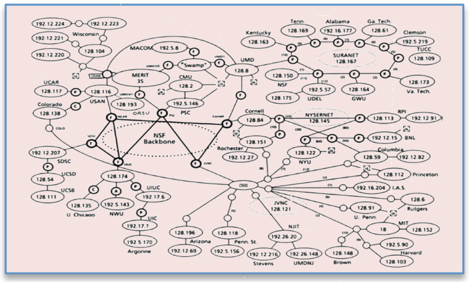








INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2012
Lecture #5

- Error correction for linear block codes
  - Syndrome decoding
- Burst errors and interleaving

Matrix Notation for Linear Block Codes

Task: given k-bit message, compute n-bit codeword. We can use standard matrix arithmetic (modulo 2) to do the job. For example, here's how we would describe the (9,4,4) rectangular code that includes an overall parity bit.

$$D \cdot G = C$$

$$\begin{bmatrix} D_1 & D_2 & D_3 & D_4 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} D_1 & D_2 & D_3 & D_4 & P_1 & P_2 & P_3 & P_4 & P_5 \end{bmatrix}$$

1×k
message
vector

k×n
generator
matrix

1×n
code word vector
in the **row space** of G

The generator matrix, $G_{kxn} = \left[\; I_{k\times k} \; \middle| \; A_{k\times(n-k)} \; \right]$

A closer look at the Parity Check Matrix A

Parity equation $$P_j = \sum_{i=1}^{k} D_i a_{ij}$$

Parity relation $$P_j + \sum_{i=1}^{k} D_i a_{ij} = 0$$

$$A = [a_{ij}]$$

So entry a_{ij} in i-th row, j-th column of A specifies whether data bit D_i is used in constructing parity bit P_j

Questions: Can two columns of A be the same? Should two columns of A be the same? How about rows?

Parity Check Matrix

Can restate the codeword generation process as a parity check or **nullspace** check

$$C.\ H^T = 0$$

$$H_{(n-k)xn} \cdot C^T_{1xn} = 0$$

The parity check matrix,

$$H = A^T | I_{(n-k)\times(n-k)}$$

For (9,4,4) example

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix} = 0_{5x1}$$

(n-k) x n
parity check
matrix

n×1
code word
vector
(transpose)

Extracting d from H

- Claim: The minimum distance d of the code C is the minimum number of columns of H that are linearly dependent, i.e., that can be combined to give the zero vector

- Proof: d = minimum-weight nonzero codeword in C

- One consequence: If A has two identical rows, then A^T has two identical columns, which means d is no greater than 2, so error correction is not possible.

Simple-minded Decoding

- Compare received n-bit word R = C + E against each of 2^k valid codewords to see which one is HD 1 away
- Doesn't exploit the nice linear structure of the code!

Syndrome Decoding – Matrix Form

Task: given n-bit code word, compute (n-k) syndrome bits. Again we can use matrix multiply to do the job.

received word $$R = C + E$$

compute Syndromes on receive word $$H \cdot R^T = S$$

(n-k) x 1 syndrome vector

To figure out the relationship of Syndromes to errors:

$$H \cdot (C + E)^T = S$$ use $$H \cdot C^T = 0$$

$$H \cdot E^T = S$$ figure-out error type from Syndrome

Knowing the error patterns we want to correct for, we can compute k Syndrome vectoroffline (or n, if you want to correct errors in the parity bits, but this is not needed) and then do a lookup after the Syndrome is calculated from a received word to find the error type that occurred

Syndrome Decoding – Steps

Step 1: For a given code and error patterns E_i, precompute Syndromes and store them

$$H \cdot E_i = S_i$$

Step 2: For each received word, compute the Syndrome

$$H \cdot R = S$$

Step 3: Find *l* such that S_l == S and apply correction for error E_l

$$C = R + E_l$$

Syndrome Decoding – Steps (9,4,4) example

Codeword generation:

$$\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1\\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1\\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1\\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

Received word in error:generation:

$$\begin{bmatrix}1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

Syndrome computation for received word

$$\begin{bmatrix}1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0\\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0\\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}1\\0\\1\\1\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\1\\1\end{bmatrix}$$

Precomputed Syndrome for a given error pattern

$$\begin{bmatrix}1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0\\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0\\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\1\\1\end{bmatrix}$$

Syndrome Decoding - Steps (9,4,4) example

Correction:

Since received word Syndrome $[1\ 0\ 0\ 1\ 1]^T$ matches the Syndrome of the error [0 1 0 0 0 0 0 0 0], apply this error to the received word to recover the original codeword

Received word

$$\lfloor 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0 \rfloor = \lfloor 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0 \rfloor + \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Corrected codeword

Error pattern from matching Syndrome

Linear Block Codes: Wrap-Up

- (n,k,d) codes have rate k/n and can correct up to floor((d-1)/2) bit errors
- Code words are linear operations over message bits: sum of any two code words is a code word
  - Message + 1 parity bit: (n+1,n,2) code
    - Good code rate, but only 1-bit error detection
  - Replicating each bit c times is a (c,1,c) code
    - Simple way to get great error correction; poor code rate
  - Hamming single-error correcting codes are (n, n-m, 3) where $n = 2^m - 1$ for $m > 1$
    - Adding an overall parity bit makes the code (n+1,n-m,4)
  - Rectangular parity codes are (rc+r+c, rc, 3) codes
    - Rate not as good as Hamming codes
- Syndrome decoding: general efficient approach for decoding linear block codes

Burst Errors

- Correcting single-bit errors is good
- Similar ideas could be used to correct independent multi-bit errors
- But in many situations errors come in bursts: correlated multi-bit errors (e.g., fading or burst of interference on wireless channel, damage to storage media etc.). How does single-bit error correction help with that?

Independent multi-bit errors

e.g., m errors in n bits

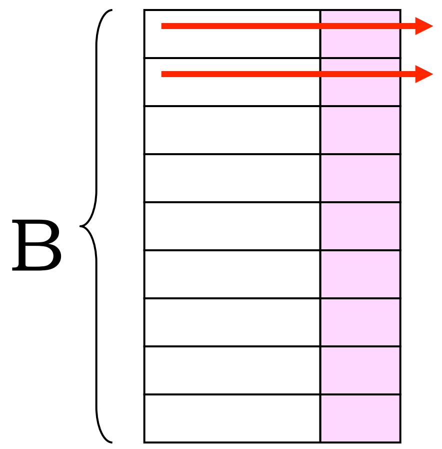
$$\binom{n}{m} p^m (1-p)^{n-m}$$

$$\binom{n}{m} = \frac{n!}{(n-m)!(m)!}$$

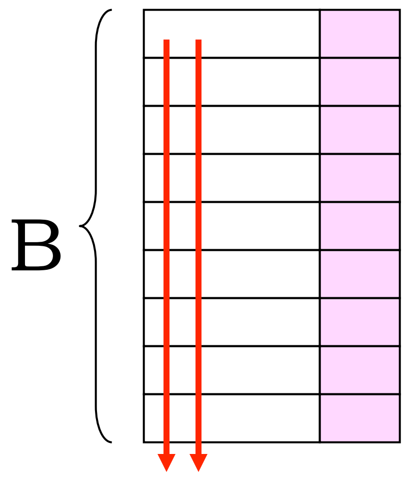
$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Coping with Burst Errors by Interleaving

Well, can we think of a way to turn a B-bit error burst into B single-bit errors?



Row-by-row transmission order



Col-by-col transmission order

Problem: Bits from a particular codeword are transmitted sequentially, so a B-bit burst produces multi-bit errors.

Solution: interleave bits from B different codewords. Now a B-bit burst produces 1-bit errors in B different codewords.

Framing

- Looking at a received bit stream, how do we know where a block of interleaved codewords begins?
- Physical indication (transmitter turns on, beginning of disk sector, separate control channel)
- Place a unique bit pattern (frame sync sequence) in the bit stream to mark start of a block
  - Frame = sync pattern + interleaved code word block
  - Search for sync pattern in bit stream to find start of frame
  - Bit pattern can't appear elsewhere in frame (otherwise our search will get confused), so have to make sure no legal combination of codeword bits can accidentally generate the sync pattern (can be tricky…)
  - Sync pattern can't be protected by ECC, so errors may cause us to lose a frame every now and then, a problem that will need to be addressed at some higher level of the communication protocol.

Summary: example channel coding steps

1. Break message stream into k-bit blocks.
2. Add redundant info in the form of (n-k) parity bits to form n-bit codeword. Goal: choose parity bits so we can correct single-bit errors.
3. Interleave bits from a group of B codewords to protect against B-bit burst errors.
4. Add unique pattern of bits to start of each interleaved codeword block so receiver can tell how to extract blocks from received bitstream.
5. Send new (longer) bitstream to transmitter.

```
011011101101
```

$\downarrow$ Step 1: k=4

```
0110
1110
1101
```

$\downarrow$ Step 2: (8,4,3) code

```
01101111
11100101
11010110
```

$\downarrow$ Step 3: B = 3

```
011111110001100111101110
```

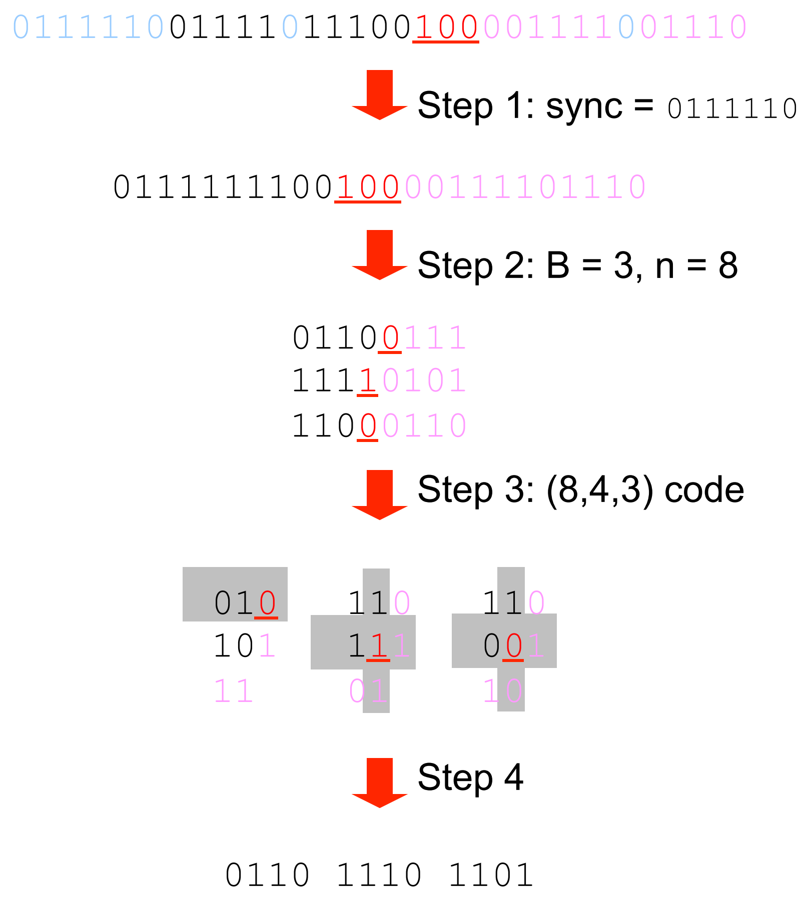
$\downarrow$ Step 4: sync = `0111110`

```
011111001111011100011001111001110
```

Sync pattern has five consecutive 1's. To prevent sync from appearing in message, "bit-stuff" 0's after any sequence of four 1's in the message. This step is easily reversed at receiver (just remove 0 after any sequence of four consecutive 1's in the message).

Summary: example error correction steps

1. Search through received bit stream for sync pattern, extract interleaved codeword block
2. De-interleave the bits to form B n-bit codewords
3. Check parity bits in each code word to see if an error has occurred. If there's a single-bit error, correct it.
4. Extract k message bits from each corrected codeword and concatenate to form message stream.


011111001111011100100001111001110
Step 1: sync = 0111110
0111111100100001111101110
Step 2: B = 3, n = 8
01100111
11110101
11000110
Step 3: (8,4,3) code
010 110 110
101 111 001
11 01 10
Step 4
0110 1110 1101