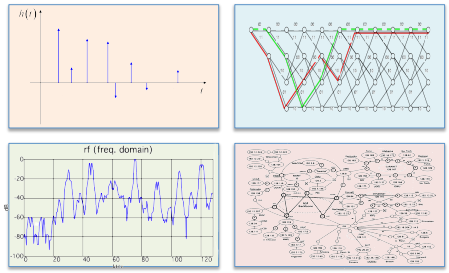


INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2009
Lecture #9

• Reed-Solomon Codes

In search of a better code

- Problem: information about a particular message unit (bit, byte, ..) is captured in just a few locations, i.e., the message unit and some number of parity units. So a small but unfortunate set of errors might wipe out all the locations where that info resides, causing us to lose the original message unit.
- Potential Solution: figure out a way to spread the info in each message unit throughout all the code words in a block. Require only some fraction good code words to recover the original message.

Thought experiment…

- Suppose you had two 8-bit values to communicate: A, B
- We'd like an encoding scheme where each transmitted value included information about both A and B
  - How about sending Ax + B for various values of x?
  - Standardize on a particular sequence for x, known to both the transmitter and receiver. That way, we don't have to actually send the x's – the receiver will know what they are. For example, x = 1, 2, 3, 4, …
  - How many values do you need to solve for A and B?
  - We'll send extra to provide for recovery from errors…

Example

- Suppose you received four values from the transmitter, corresponding to x = 1, 2, 3 and 4
  - 73, 249, 321, 393
- We need a pair of values to solve for A and B; there are six possible pairs chosen from four values
  - (73,249) (73,321) (73,393) (249,321) (249,393) (321,393)
- Take each pair and solve for A and B

$A \cdot 1 + B = 73$	$A \cdot 1 + B = 73$	$A \cdot 1 + B = 73$
$A \cdot 2 + B = 249$	$A \cdot 3 + B = 321$	$A \cdot 4 + B = 393$
A=175, B=-102	*A=124, B=-51*	*A=106.6, B=-33.6*
$A \cdot 2 + B = 249$	$A \cdot 2 + B = 249$	$A \cdot 3 + B = 321$
$A \cdot 3 + B = 321$	$A \cdot 4 + B = 393$	$A \cdot 4 + B = 393$
A=72, B=105	*A=72, B=105*	*A=72, B=105*

- Majority rules: A=72, B=105 (the 73 was an error)

Spreading the wealth…

- Idea: oversampled polynomials. Let

  $P(x) = m_0 + m_1x + m_2x^2 + \ldots + m_{k-1}x^{k-1}$

  where $m_0, m_1, \ldots, m_{k-1}$ are the *k* message units to be encoded. Transmit value of polynomial at *n* different predetermined points $v_0, v_1, \ldots, v_{n-1}$:

  $P(v_0), P(v_1), P(v_2), \ldots, P(v_{n-1})$

  Use any *k* of the received values to construct a linear system of *k* equations which can then be solved for *k* unknowns $m_0, m_1, \ldots, m_{k-1}$. Each transmitted value contains info about all m_i.
- Note that using integer arithmetic, the P(v) values are numerically greater than the m_i and so require more bits to represent than the m_i. In general the encoded message would require a lot more bits to send than the original message!

Solving for the m_i

- Solving k *linearly independent* equations for the k unknowns (i.e., the m_i):

$$\begin{pmatrix} 1 & v_0 & v_0^2 & \cdots & v_0^{k-1} \\ 1 & v_1 & v_1^2 & \cdots & v_1^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_{k-1} & v_{k-1}^2 & \cdots & v_{k-1}^{k-1} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{k-1} \end{pmatrix} = \begin{pmatrix} P(v_0) \\ P(v_1) \\ \vdots \\ P(v_{k-1}) \end{pmatrix}$$

- Solving a set of linear equations using Gaussian Elimination (multiplying rows, switching rows, adding multiples of rows to other rows) requires add, subtract, multiply and divide operations.
- These operations (in particular division) are only well defined over *fields,* e.g., rational numbers, real numbers, complex numbers -- not at all convenient to implement in hardware.

Finite Fields to the Rescue

- Reed's & Solomon's idea: do all the arithmetic using a finite field (also called a Galois field). If the m_i have B bits, then use a finite field with order 2^B so that there will be a field element corresponding to each possible value for m_i.
- For example with B = 2, here are the tables for the various arithmetic operations for a finite field with 4 elements. Note that every operation yields an element in the field, i.e., the result is the same size as the operands.

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

A	-A	A^{-1}
0	0	0
1	1	1
2	2	3
3	3	2

A + (-A) = 0 A * (A^{-1}) = 1

How many values to send?

- Note that in a Galois field of order 2^B there are at most 2^B unique values v we can use to generate the P(v)
  - if we send more than 2^B values, some of the equations we might use when solving for the m_i may not be linearly independent and we won't have enough information to find a unique solution for the m_i.
  - Sending P(0) isn't very interesting (only involves m_0)
- Reed-Solomon codes use $n = 2^B-1$ (n is the number of P(v) values we generate and send).
  - For many applications B = 8, so n = 255
  - A popular R-S code is (255,223), i.e., a code block consisting of 223 8-bit data bytes + 32 check bytes

Use for error correction

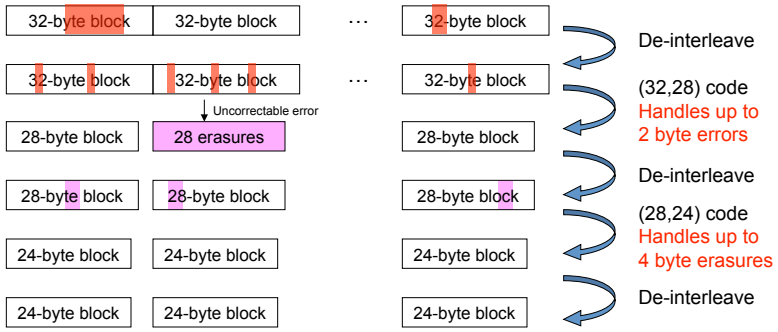
- If one of the $P(v_i)$ is received incorrectly, if it's used to solve for the m_i, we'll get the wrong result.
- So try all possible (n choose k) subsets of values and use each subset to solve for m_i. Choose solution set that gets the majority of votes.
  - No winner? Uncorrectable error… throw away block.
- (n,k) code can correct up to (n-k)/2 errors since we need enough good values to ensure that the correct solution set gets a majority of the votes.
  - R-S (255,223) code can correct up to 16 symbol errors; good for error bursts: 16 consecutive symbols = 128 bits!

Erasures are special

- If a particular received value is known to be erroneous (an "erasure"), don't use it all!
  - How to tell when received value is erroneous? Sometimes there's channel information, e.g., carrier disappears.
  - See next slide for clever idea based on concatenated R-S codes
- (n,k) R-S code can correct n-k erasures since we only need k equations to solve for the k unknowns.
- Any combination of E errors and S erasures can be corrected so long as $2E + S \leq n-k$.

Example: CD error correction

- On a CD: two concatenated R-S codes



Result: correct up to 3500-bit error bursts (2.4mm on CD surface)