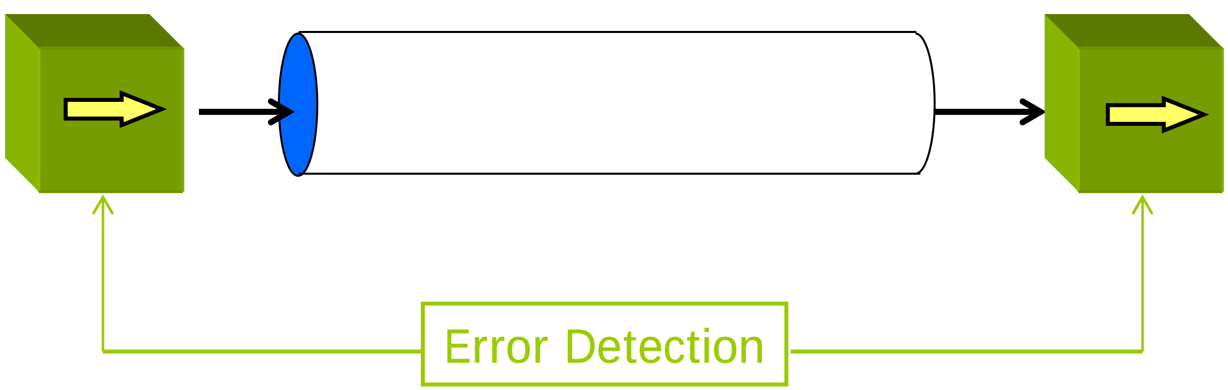
Error Detection and Correction

CS 571

Fall 2006

The Problem

- Given a frame channel that may deliver frames with corrupted symbols (flipped bits)
- How do we determine
  - Whether a received frame was corrupted in transit?
  - If so, what was the original transmitted frame?
- So: design protocol boxes to detect corruption



The Solution

<u>The Fundamental Paradigm of Error Detection</u>:

- Sender and Receiver agree (in advance) that all transmitted frames shall have a particular property
- Sender converts every data frame to one having the property before transmitting
- Receiver discards any received frame that does not have the property

- What's left to do?
  - Come up with a suitable property!

When thinking about error detection mechanisms, ask yourself: What is the <u>property</u>?

The Solution

Characteristics of a "good" property:

- Easy to convert an arbitrary frame to one with the property
  - Generally do this by adding some specially-chosen bits to the frame
- Easy to check whether a frame has the property
  - Do we expect this to be implemented in hardware? software?
- Errors are <u>unlikely</u> to preserve the property

- Observation:
  - The last characteristic depends entirely on
    - The fraction of all possible frames that have the property
    - The distribution of errors (i.e. likely received frames)

Example: Parity

- Property: transmitted frames all contain an even number of 1's
  - Receiver discards frames with an odd number of 1's
- Is this a good property?
  - ✓ Easy to make a frame have it (add one bit)
  - ✓ Easy to check if frame has it (simple FSM)
  - o Errors unlikely to preserve the property?
    - We can't say without some knowledge of error probabilities!

Thinking About Error Distributions

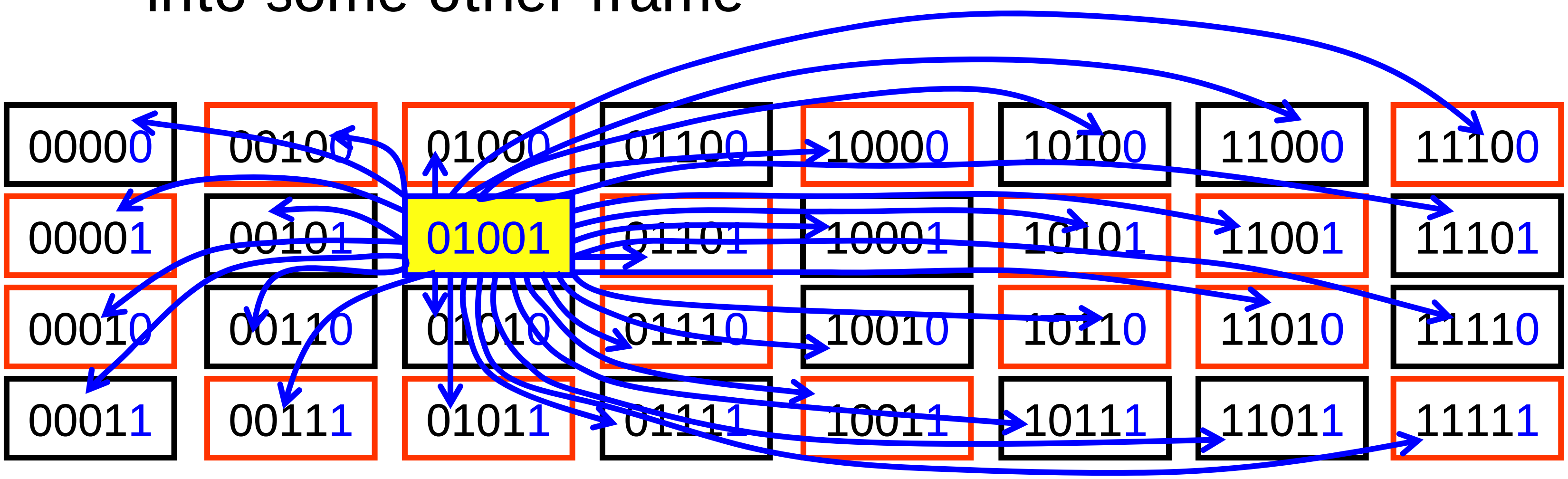
- Until further notice: assume fixed frame sizes
- Consider a channel that transmits 5-bit frames
  - Use 4 as data bits, one as parity
- There are 32 possible 5-bit frames:

00000	00100	01000	01100	10000	10100	11000	11100
00001	00101	01001	01101	10001	10101	11001	11101
00010	00110	01010	01110	10010	10110	11010	11110
00011	00111	01011	01111	10011	10111	11011	11111

- Half have incorrect parity, and will never be sent

Thinking About Error Distributions

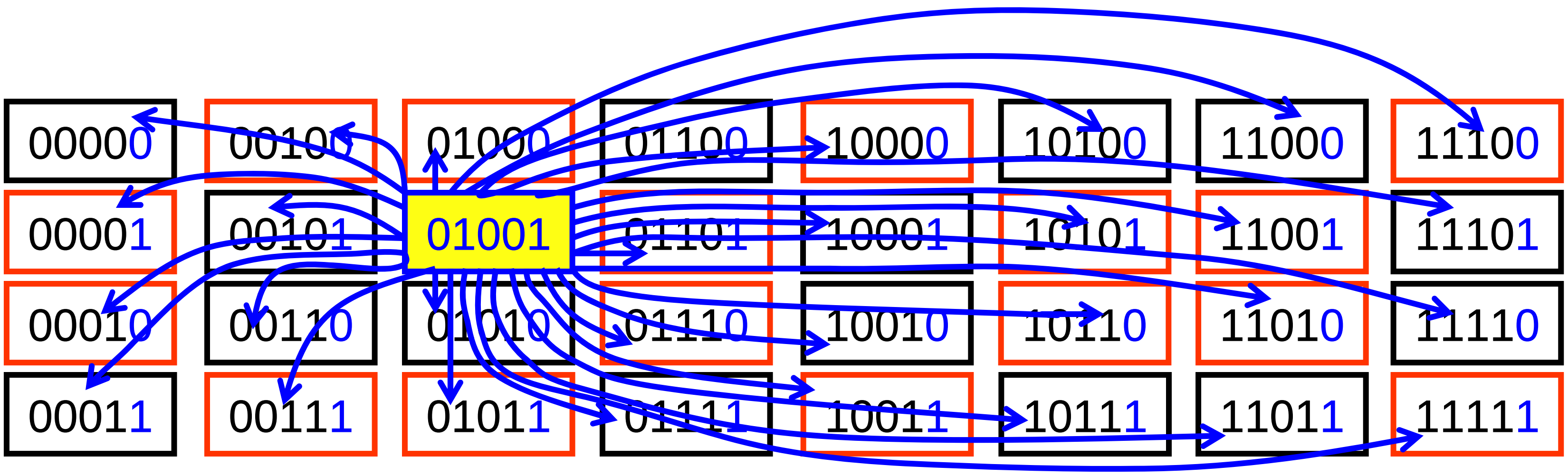
- Assume that <u>any error is equally likely</u>, given than an error occurs
  - That is, a frame emerges unchanged with some probability *p*, while with probability 1-*p* it is changed into some other frame



- Of the possible received corrupted frames:
  - 15 have correct parity
  - 16 have incorrect parity

Thinking About Error Distributions

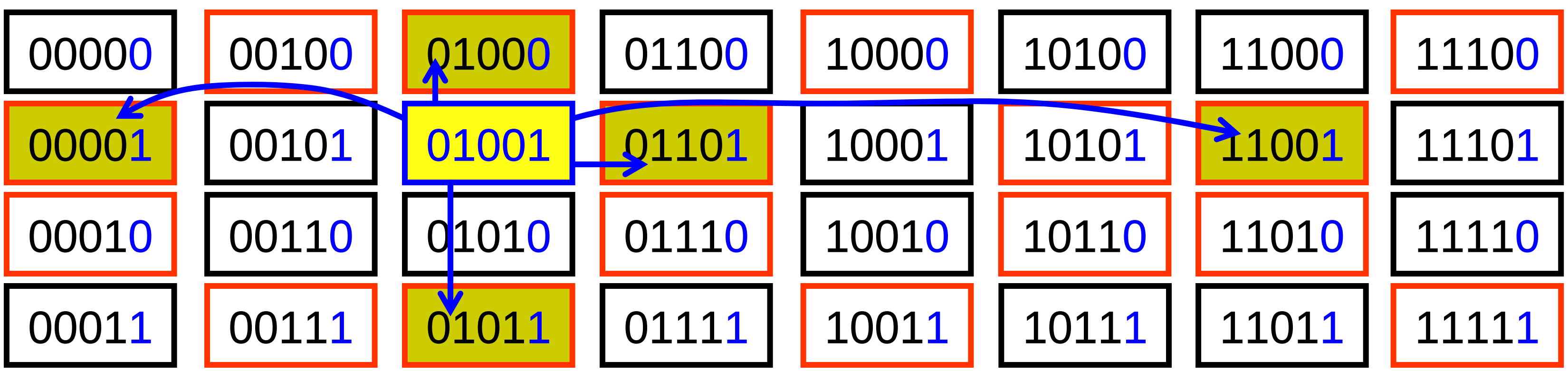
- The probability that an error is detected (given than one occurs) is just 0.5!
- Not a very good detector



- Of the possible received corrupted frames:
  - 15 have correct parity
  - 16 have incorrect parity

Thinking About Error Distributions

- On the other hand, if single-bit errors are much more likely than others, it is a good detector!



Hamming Distance

- Given two n-bit frames ("words") w and w'
- Hamming Distance HD(w,w') between them is
  - the number of bit positions in which they differ, or
  - the number of ones in their bitwise-sum (mod 2)
- Examples:

  HD(1001,0101) = 2

  HD(101,010) = 3

  HD(100,000) = 1

Error Distribution Models

- For most real channels, the probability of an error transforming a transmitted frame w into a received frame w' decreases with increasing HD(w,w')
- This is true for Binary Symmetric Channels:
  - Each bit is corrupted with fixed probability p
    - Typically $p << 1/2$
  - Errors are independent:
    - If one bit is corrupted, the probability of bits before/after it does not change
  - Probability k (of n) bits corrupted = $C(n,k)p^k(1-p)^{n-k}$

Error Distribution Models

- Binary Symmetric Channel often does not model real channels adequately
- In real channels, errors tend to occur in <u>bursts</u>
  - The probability of a bit being corrupted increases if other bits "near" it are corrupted
  - That is, errors are <u>not independent</u>
- For these channels, the important characteristic is the <u>burst length distribution</u>
  - That is, the probability of a burst error of length k, for k=1, 2, …
- For most channels, longer bursts <u>are</u> less likely

More Sophisticated Parity Codes

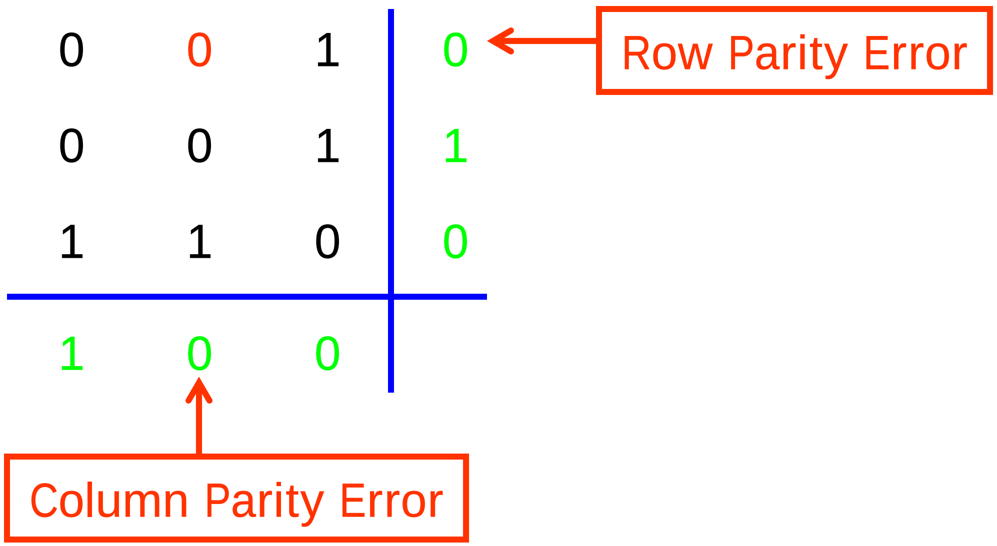
- Idea: arrange data bits in a square array
- Add a parity bit for each row and column
  - Each data bit is "protected" by two parity bits

d_0	d_1	d_2	r_3
d_3	d_4	d_5	r_4
d_6	d_7	d_8	r_5
r_0	r_1	r_2	

0	1	1	0
0	0	1	1
1	1	0	0
1	0	0	

More Sophisticated Parity Codes

- This code can detect and correct all one-bit errors
  - Row and column parity errors point to the corrupted data bit
  - If a parity bit is corrupted, other parity bits are OK
  - Note implicit assumption: single-bit error is most likely!



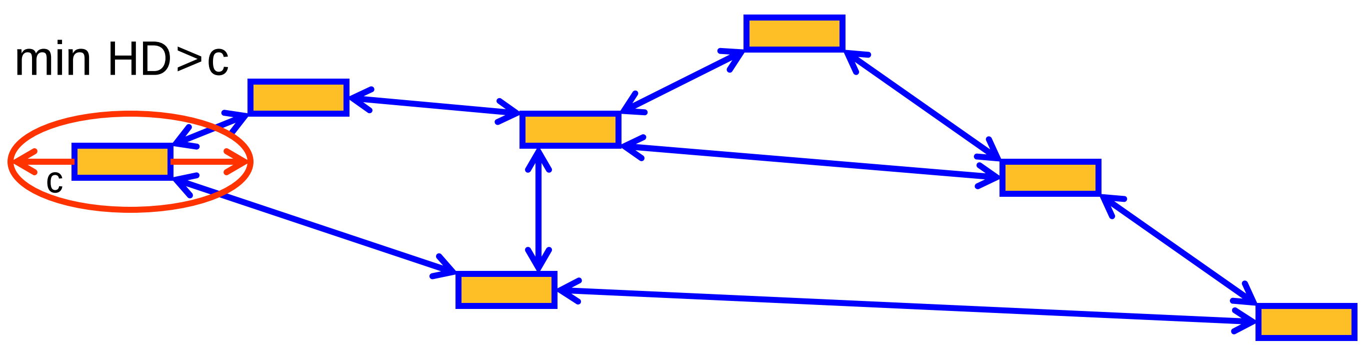
More Sophisticated Parity Codes

- This code can detect all two-bit errors
- Can't correct, due to ambiguity

0	0	1	0
0	0	1	1
1	1	1	0
1	0	0	

General Result: Error detection capabilities

- Code: subset of $\{0,1\}^n$ (binary strings of length n)
- If minimum Hamming Distance between two code words exceeds d, the code can detect up to d errors
  - More errors required to transform one word into another



General Result: Error detection capabilities

- Theorem: For codes based on even parity, the minimum distance between two code words is equal to the <u>minimum weight</u> of a nonzero code word
  - Weight of a code word = # 1's in the word
- For row-column parity: min weight = 3

0	0	0	0
0	0	0	0
0	0	1	1
0	0	1	1

We can easily increase it to 4 by adding a "parity of parities" bit!

Checksums

- Property: when the frame is viewed as a sequence of (fixed-size) words, the sum of those words is some predetermined value
  - Different kinds of arithmetic may be used!
- Good property?
  - Easy to establish: just add a word
  - Easy to check: add up the words of the received frame
  - Unlikely to be preserved?
    - As always, depends on the error distribution

The "Internet Checksum"

- Checksum used in IP, TCP and UDP
- Designed to be easily implementable in software

Cyclic Redundancy Check (CRC)

- Used in many modern datalink protocols
- Excellent error detection capabilities
  - An r-bit CRC can detect all but $1/2^r$ error patterns
    - Assumption: bursts are limited in length
  - Example: CRC-16
    - Detects all error bursts up to 16 bits in length
    - Detects all errors with odd weight
    - Detects all but $1/2^{16}$ of bursts longer than 16 bits

CRC: Background

- Basic concept: Strings of bits viewed as polynomials over the finite field F_2
  - Each coefficient is either 0 or 1
    - Example: 101101 corresponds to $x^5 + x^3 + x^2 + 1$
    - k-bit string corresponds to a polynomial of degree k-1
  - Addition of polynomials: modulo-2 addition of coefficients of like terms
    - Subtraction: same as addition
    - Example: 101101 $\leftrightarrow$ $x^5 + x^3 + x^2 + 1$
      + 1001001 $\leftrightarrow$ $x^6 + x^3 + 1$
      = 1100100 $\leftrightarrow$ $x^6 + x^5 + x^2 + 1$
  - Multiplication as in normal algebra
    - Example: $(x^3 + x^2 + 1)(x^2 + 1) = x^5 + x^4 + x^3 + 1$

CRC: How It Works

- Sender and Receiver agree in advance on a Generator Polynomial G(x) of degree r
- The Property: divisibility by G(x)
  - Every transmitted frame, when viewed as a polynomial T(x), is a multiple of the polynomial G(x)
  - Receiver divides every received frame by G(x), discards any that have nonzero remainder
- Easily implemented in hardware using shift regs
- Easy to make any frame (of any length!) correspond to a multiple of G(x)

CRC: Sender Procedure

To transmit $W(x)$:

1. Multiply $W(x)$ by x^r

   This shifts it left r bits, adding 0's on the right

2. Divide $W(x)x^r$ by $G(x)$. Let remainder = $R(x)$

   The degree of $R(x)$ is less than r

3. Transmit $T(X) = W(x)x^r + R(x)$

   Because addition = subtraction, $T(x)$ has remainder 0 when divided by $G(x)$

Sender Procedure Example

Assume G(x) = 1101 $\leftrightarrow$ $x^3 + x^2 + 1$

To transmit W(x) = 111010:

1. Multiply by x^r to get 111010000
2. Divide $W(x)x^r$ by G(x) — ignore quotient!

```
     ___________
1101 )111010000
      1101
        1110
        1101
          1100
          1101
           010   <-- Remainder R(x)
```

3. Transmit $W(x)x^r$ + R(x) = 111010010

CRC: Receiver Procedure

- Received frame = D(x)
- Divide D(x) by G(x)
- If remainder = 0, accept; else discard
- Example: D(x) = 111010010

```
1101 ) 111010010
       1101
         1110
         1101
           1101
           1101
              00
```

Remainder = 0 ⇒ accept!

CRC: Why Is This a Good Property?

- Under what conditions is the property "divisibility by G(x)" preserved?
- View the channel as adding an error polynomial E(x) to the transmitted frame T(x)
  - Received frame D(x) = T(x) + E(x)
  - D(x) is divisible by G(x) iff E(x) is!
- Choose G(x) unlikely to be divisible by E(x)
  - Example: G(x) has a constant term $\Rightarrow$ catch single-bit errors
  - Given some knowledge of error distribution, can design G(x) to catch all errors shorter than some burst length
    - Typically less than 2^{-r} of all possible errors go undetected