• Some bits may be received in error due to noise. How do we detect this?
  – Parity »
  – Checksums »
  – CRCs »

• Detection will let us fix the error, for example, by retransmission (later).
Simple Error Detection – Parity Bit

• Take D data bits, add 1 check bit that is the sum of the D bits
  – Sum is modulo 2 or XOR
Parity Bit (2)

- How well does parity work?
  - What is the distance of the code?
  - How many errors will it detect/correct?

- What about larger errors?
Checksums

• Idea: sum up data in N-bit words
  – Widely used in, e.g., TCP/IP/UDP

  1500 bytes  16 bits

• Stronger protection than parity
Internet Checksum

• Sum is defined in 1s complement arithmetic (must add back carries)
  – And it’s the negative sum

• “The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words ...” – RFC 791
Internet Checksum (2)

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

0001
f203
f4f5
f6f7
Internet Checksum (3)

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

\[
\begin{array}{c}
0001 \\
f203 \\
f4f5 \\
f6f7 \\
+(0000) \\
\hline
2ddf0 \\
\downarrow \\
ddf0 \\
+ \\
2 \\
\hline \\
ddf2 \\
\downarrow \\
220d
\end{array}
\]
Internet Checksum (4)

Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0
Internet Checksum (5)

Receiving:
1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0
Internet Checksum (6)

• How well does the checksum work?
  – What is the distance of the code?
  – How many errors will it detect/correct?

• What about larger errors?
Cyclic Redundancy Check (CRC)

• Even stronger protection
  – Given n data bits, generate k check bits such that the n+k bits are evenly divisible by a generator C

• Example with numbers:
  – n = 302, k = one digit, C = 3
CRCs (2)

• The catch:
  – It’s based on mathematics of finite fields, in which “numbers” represent polynomials
  – e.g., 10011010 is $x^7 + x^4 + x^3 + x^1$

• What this means:
  – We work with binary values and operate using modulo 2 arithmetic
CRCs (3)

• Send Procedure:
  1. Extend the n data bits with k zeros
  2. Divide by the generator value C
  3. Keep remainder, ignore quotient
  4. Adjust k check bits by remainder

• Receive Procedure:
  1. Divide and check for zero remainder
Data bits: 11010111111
Check bits: 
\( C(x) = x^4 + x^1 + 1 \)
\( C = 10011 \)
\( k = 4 \)
CRCs (5)

Transmitted frame: 1101011111110010

Frame with four zeros appended minus remainder: 0010

Quotient (thrown away): 110000111110

Frame with four zeros appended: 000000
CRCs (6)

• Protection depend on generator
  – Standard CRC-32 is 10000010
    01100000 10001110 110110111

• Properties:
  – HD=4, detects up to triple bit errors
  – Also odd number of errors
  – And bursts of up to k bits in error
  – Not vulnerable to systematic errors like checksums
Error Detection in Practice

• CRCs are widely used on links
  – Ethernet, 802.11, ADSL, Cable …

• Checksum used in Internet
  – IP, TCP, UDP … but it is weak

• Parity
  – Is little used
Topic

- Some bits may be received in error due to noise. How do we fix them?
  - Hamming code
  - Other codes

- And why should we use detection when we can use correction?
Why Error Correction is Hard

- If we had reliable check bits we could use them to narrow down the position of the error
  - Then correction would be easy
- But error could be in the check bits as well as the data bits!
  - Data might even be correct
Intuition for Error Correcting Code

• Suppose we construct a code with a Hamming distance of at least 3
  – Need ≥3 bit errors to change one valid codeword into another
  – Single bit errors will be closest to a unique valid codeword

• If we assume errors are only 1 bit, we can correct them by mapping an error to the closest valid codeword
  Works for d errors if HD ≥ 2d + 1
Intuition (2)

- Visualization of code:

Valid codeword

Error codeword
Intuition (3)

- Visualization of code:

  - Single bit error from A
  - Three bit errors to get to B
  - Valid codeword
  - Error codeword
Hamming Code

• Gives a method for constructing a code with a distance of 3
  
  → Uses $n = 2^k - k - 1$, e.g., $n=4$, $k=3$
    
    – Put check bits in positions $p$ that are powers of 2, starting with position 1
    
    – Check bit in position $p$ is parity of positions with a $p$ term in their values

• Plus an easy way to correct [soon]
Hamming Code (2)

- Example: data=0101, 3 check bits
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7
  - Check 2 covers positions 2, 3, 6, 7
  - Check 4 covers positions 4, 5, 6, 7

\[ p_1 = 0 + 1 + 1 = 0 \]
\[ p_2 = 0 + 0 + 1 = 1 \]
\[ p_3 = 1 + 0 + 0 = 1 \]

0 1 0 0 1 0 1
1 2 3 4 5 6 7
Hamming Code (3)

- Example: data=0101, 3 check bits
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7
  - Check 2 covers positions 2, 3, 6, 7
  - Check 4 covers positions 4, 5, 6, 7

\[
\begin{array}{cccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
\]

\[p_1 = 0+1+1 = 0, \quad p_2 = 0+0+1 = 1, \quad p_4 = 1+0+1 = 0\]
Hamming Code (4)

• To decode:
  – Recompute check bits (with parity sum including the check bit)
  – Arrange as a binary number
  – Value (syndrome) tells error position
  – Value of zero means no error
  – Otherwise, flip bit to correct
Hamming Code (5)

- Example, continued

\[
\begin{array}{ccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\[\begin{align*}
p_1 &= 0 + 0 + 1 = 0 \\
p_2 &= 1 + 0 + 1 = 0 \\
p_4 &= 0 + 1 + 0 = 0 \\
\end{align*}\]

Syndrome = 000 (Corrected)

Data = 0 1 0 1
Hamming Code (6)

• Example, continued

$$\rightarrow \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
\end{array}$$

\[ p_1 = 0+0+1+1 = 0, \quad p_2 = 1+0+0+1 = 0, \]
\[ p_4 = 0+1+0+1 = 0 \]

Syndrome = 000, no error
Data = 0 1 0 1
Hamming Code (7)

- Example, continued

\[ \rightarrow 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \]

\[
\begin{align*}
1 & \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \\
p_1 &= 0 + 0 + 1 + 0 = 1 \\
p_2 &= 1 + 0 + 1 + 1 = 1 \\
p_4 &= 0 + 1 + 1 + 1 = 1
\end{align*}
\]

Syndrome = \[1 \ 1 \ 0 \ \rightarrow \ 6 \]

Data = \[0 \ 1 \ 0 \ 1 \ 0 \ 1 \]
Hamming Code (8)

- Example, continued

\[ \begin{array}{cccccccc}
& 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\text{1} & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} \]

\[ p_1 = 0 + 0 + 1 + 1 = 0, \quad p_2 = 1 + 0 + 1 + 1 = 1, \]
\[ p_4 = 0 + 1 + 1 + 1 = 1 \]

Syndrome = 1 1 0, flip position 6
Data = 0 1 0 1 (correct after flip!)
Other Error Correction Codes

• Codes used in practice are much more involved than Hamming

• Convolutional codes (§3.2.3)
  – Take a stream of data and output a mix of the recent input bits
  – Makes each output bit less fragile
  – Decode using Viterbi algorithm (which can use bit confidence values)
Other Codes (2) – LDPC

• Low Density Parity Check (§3.2.3)
  – LDPC based on sparse matrices
  – Decoded iteratively using a belief propagation algorithm
  – State of the art today

• Invented by Robert Gallager in 1963 as part of his PhD thesis
  – Promptly forgotten until 1996 ...
Detection vs. Correction

• Which is better will depend on the pattern of errors. For example:
  – 1000 bit messages with a bit error rate (BER) of 1 in 10000

• Which has less overhead?
Detection vs. Correction

• Which is better will depend on the pattern of errors. For example:
  – 1000 bit messages with a bit error rate (BER) of 1 in 10000

• Which has less overhead?
  – It still depends! We need to know more about the errors
Detection vs. Correction (2)

1. Assume bit errors are random
   - Messages have 0 or maybe 1 error

• Error correction:
   - Need ~10 check bits per message
   - Overhead: $10$

• Error detection:
   - Need ~1 check bits per message plus 1000 bit retransmission 1/10 of the time
   - Overhead: $1 + \frac{1000}{10} \approx 101$ bits
Detection vs. Correction (3)

2. Assume errors come in bursts of 100
   - Only 1 or 2 messages in 1000 have errors

   • Error correction:
     - Need >>100 check bits per message
     - Overhead: >100?

   • Error detection:
     - Need 32? check bits per message plus 1000 bit resend 2/1000 of the time
     - Overhead: \(32 + \frac{1000}{1000} = 34\) bits
Detection vs. Correction (4)

• Error correction:
  – Needed when errors are expected
  – Or when no time for retransmission

• Error detection:
  – More efficient when errors are not expected
  – And when errors are large when they do occur
Error Correction in Practice

- Heavily used in physical layer
  - LDPC is the future, used for demanding links like 802.11, DVB, WiMAX, LTE, power-line, ...  
  - Convolutional codes widely used in practice

- Error detection (w/ retransmission) is used in the link layer and above for residual errors

- Correction also used in the application layer
  - Called Forward Error Correction (FEC)  
  - Normally with an erasure error model
  - E.g., Reed-Solomon (CDs, DVDs, etc.)