Error Detection

- 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
Internet Checksum (3)

Sending:

1. Arrange data in 16-bit words

2. Put zero in checksum position, add

\[
\begin{array}{c}
\text{0001} \\
\text{f203} \\
\text{f4f5} \\
\text{f6f7}
\end{array}
\]
\[+(0000)\]
\[
\begin{array}{c}
\text{2ddf0}
\end{array}
\]

3. Add any carryover back to get 16 bits

\[
\begin{array}{c}
\text{ddf0} \\
\text{+ 2}
\end{array}
\]
\[
\begin{array}{c}
\text{ddf2}
\end{array}
\]
\[
\begin{array}{c}
\downarrow
\end{array}
\]
\[
\begin{array}{c}
\text{220d}
\end{array}
\]

4. Negate (complement) to get sum
Internet Checksum (4)

Receiving:

1. Arrange data in 16-bit words

2. Checksum will be non-zero, add

```
 0001
f203
f4f5
f6f7
+ 220d
------
```

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, $C = 3$
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: 1101011111

Check bits:
C(x) = x^4 + x^1 + 1
C = 10011
k = 4
CRCs (5)

Transmitted frame: 1 1 0 1 0 1 1 1 1 1 1 0 0 1 0 0 0 1 0

Frame with four zeros appended minus remainder

Quotient (thrown away)

Frame with four zeros appended

Remainder
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
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