Error detection: Outline

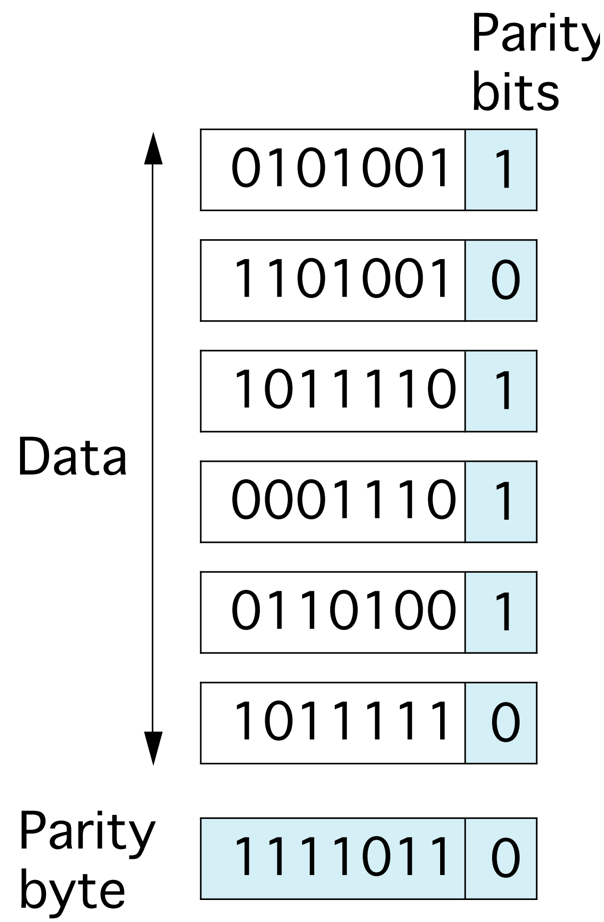
- In the last class, we looked at the problem of encoding bits on the wire and framing to delineate when a frame begins and ends.
- Today we look at how we detect errors introduced by the network
  - Mechanisms depend on how much computational overhead is tolerable, how much extra bits are wasted and what types of errors (number of errors, error types etc.) have to be detected
- First problem is to detect errors. The second problem is to correct errors
  - Correction can be achieved by resending the frame
  - Or by sending extra bits so that the receiver can reconstruct the erroneous frame

Error detection and correction.

- Parity or checksums
  - Send additional bits that can help identify if there was an error in the transmission
  - The goal is to keep this extra bits as small as possible
- One dimensional parity
  - Add one extra bit to a 7-bit code to keep either a odd- or even- number of 1s in a byte
- Two dimensional parity
  - Calculates parity across all bit (in a given bit position) in the frame
  - This uses an extra parity byte

Two dimensional parity

- It can be shown that two dimensional parity catches all 1, 2 and 3 bit errors and most 4-bit errors
- In this example, we added 14 bits of redundant information to a 42 bit message



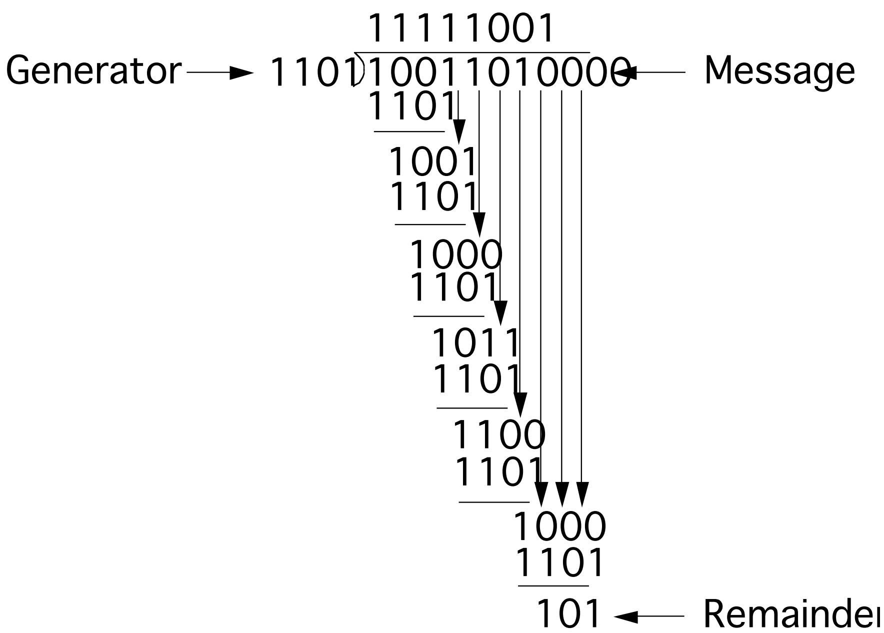
Internet checksum

- Add all the bytes and then transmit the sum.
  - Receiver does the same summation and checks the sum. If they don't match, then there was an error
- Internet checksum:
  - Consider data as 16-bit integers. Add them using 16-bit ones complement arithmetic
    - In ones complement, negative number is represented each bit inverted
    - Ones complement addition, carryout from most significant bit is added to reslt
  - Take ones complement of the result
  - Resulting 16 bit number is the checksum
- Overhead is 16 bits per message
- Internet checksum is simple but does not detect many errors - used in conjunction with others

Cyclic Redundancy Check (CRC)

- Fairly intensive computation
  - 32 bit CRC can check errors for a longer message
  - Tradeoff between computational complexity and check requirements
  - CRCs are based on finite fields
- Assume (n+1) bit message as a polynomial of degree n. Choose a CRC polynomial C(x)
  - When transmitting message M(x) of size, transmit k extra bits such that the new message P(x) is exactly divisible by C(x)
  - Receive does the same, divide P(x) with C(x). If there is no remainder, then there was no errors

- Polynomial arithmetic modulo 2
  - If polynomials of same degree, then they device
  - Subtraction is basically a xor operation
    - Xor is 1 if the two bits are different (0 & 1 or 1 & 0)
    - Consider M(x)=1001101 and C(x)=1101 and k=3



- Key is to choose C(x) such that common errors are caught
  - CRC-8: 100000111
- Each CRC function has different strengths in detecting error conditions
  - E.g. all single-bit errors, as long as xk and x0 terms have nonzero coefficient
- CRC checksums are easily implemented in hardware

Take away message

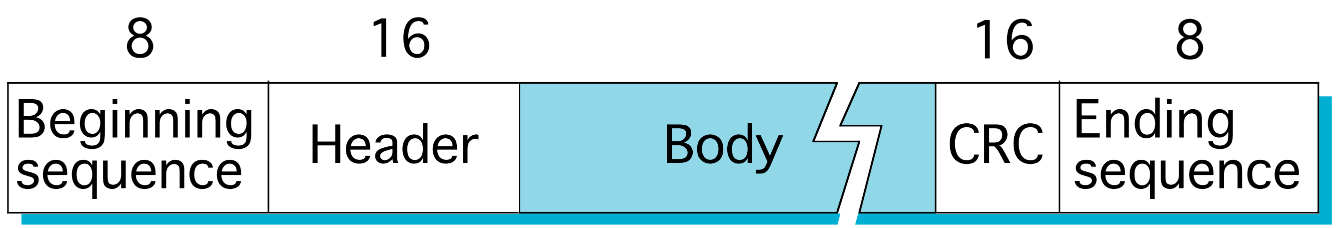
- Choosing the right error checking mechanism is a tradeoff between computational complexity and errors that you want to detect

- Multiple layers will do their own error checking, improving error detection

Questions

- What happens if the error detection mechanism did not detect a particular error?
  - Is it possible at all?

- Going back to yesterdays work:



  - What bits should the CRC cover?
  - Should CRC cover the sequence header? Sequence trailer? Why?