Error Detection

Outline
Parity
Checksum
CRC

2-Dimensional Parity

Parity bits

Data

Parity byte

Internet Checksum Algorithm

- View message as a sequence of 16-bit integers; sum using 16-bit ones-complement arithmetic; take ones-complement of the result.

```c
#include <netinet/in.h>

u_short
cksum(u_short *buf, int count)
{
    register u_long sum = 0;
    while (count--)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            /* carry occurred, so wrap around */
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```
Cyclic Redundancy Check

- Add \( k \) bits of redundant data to an \( n \)-bit message
  - want \( k << n \)
  - e.g., \( k = 32 \) and \( n = 12,000 \) (1500 bytes)
- Represent \( n \)-bit message as \( n-1 \) degree polynomial
  - e.g., MSG=10011010 as \( M(x) = x^7 + x^4 + x^3 + x^1 \)
- Let \( k \) be the degree of some divisor polynomial
  - e.g., \( C(x) = x^3 + x^2 + 1 \)

CRC (cont)

- Transmit polynomial \( P(x) \) that is evenly divisible by \( C(x) \)
  - shift left \( k \) bits, i.e., \( M(x)x^k \)
  - subtract remainder of \( M(x)x^k / C(x) \) from \( M(x)x^k \)
- Receiver polynomial \( P(x) + E(x) \)
  - \( E(x) = 0 \) implies no errors
- Divide \( (P(x) + E(x)) \) by \( C(x) \); remainder zero if:
  - \( E(x) \) was zero (no error), or
  - \( E(x) \) is exactly divisible by \( C(x) \)

Selecting \( C(x) \)

- All single-bit errors, as long as the \( x^k \) and \( x^0 \) terms have non-zero coefficients.
- All double-bit errors, as long as \( C(x) \) contains a factor with at least three terms
- Any odd number of errors, as long as \( C(x) \) contains the factor \( (x + 1) \)
- Any 'burst' error (i.e., sequence of consecutive error bits) for which the length of the burst is less than \( k \) bits.
- Most burst errors of larger than \( k \) bits can also be detected
- See Table 2.5 on page 96 for common \( C(x) \)
Stop-and-Wait

- Problem: keeping the pipe full
- Example
  - 1.5Mbps link x 45ms RTT = 67.5Kb (8KB)
  - 1KB frames implies 1/8th link utilization

Sliding Window

- Allow multiple outstanding (un-ACKed) frames
- Upper bound on un-ACKed frames, called window

SW: Sender

- Assign sequence number to each frame (SeqNum)
- Maintain three state variables:
  - send window size (SWS)
  - last acknowledgment received (LAR)
  - last frame sent (LFS)
- Maintain invariant: \( LFS - LAR \leq SWS \)
- Advance LAR when ACK arrives
- Buffer up to SWS frames
SW: Receiver

- Maintain three state variables
  - receive window size (RWS)
  - largest acceptable frame (LAF)
  - last frame received (LFR)
- Maintain invariant: LAF – LFR <= RWS

Frame SeqNum arrives:
- if LFR < SeqNum <= LAF accept
- if SeqNum < LFR or SeqNum > LAF discarded
- Send cumulative ACKs

Sequence Number Space

- SeqNum field is finite; sequence numbers wrap around
- Sequence number space must be larger than number of outstanding frames
- SWS <= MaxSeqNum - 1 is not sufficient
  - suppose 3-bit SeqNum field (0..7)
  - SWS=RWS=7
  - sender transmit frames 0..6
  - arrive successfully, but ACKs lost
  - sender retransmit 0..6
  - receiver expecting 7, 0..5, but receives second incarnation of 0..5
- SWS <= (MaxSeqNum+1) / 2 is correct rule
- Intuitively, SeqNum “slides” between two halves of sequence number space