## Point-to-Point Links

#### Outline Encoding Framing Error Detection Sliding Window Algorithm

Spring 2005

CS 461

## Problem: Consecutive 1s or 0s

- Low signal (0) may be interpreted as no signal
- High signal (1) leads to baseline wander
- Unable to recover clock

## Encoding

- Signals propagate over a physical medium
  - modulate electromagnetic waves
  - e.g., vary voltage
- Encode binary data onto signals
  - e.g., 0 as low signal and 1 as high signal
  - known as Non-Return to zero (NRZ)



## Alternative Encodings

- Non-return to Zero Inverted (NRZI)
  - make a transition from current signal to encode a one; stay at current signal to encode a zero
  - solves the problem of consecutive ones
- Manchester
  - transmit XOR of the NRZ encoded data and the clock
  - only 50% efficient (bit rate = 1/2 baud rate)

3

1

Spring 2005

## Encodings (cont)



## Encodings (cont)

### • 4B/5B

Spring 2005

- every 4 bits of data encoded in a 5-bit code
- 5-bit codes selected to have no more than one leading 0 and no more than two trailing 0s
- thus, never get more than three consecutive 0s
- resulting 5-bit codes are transmitted using NRZI
- achieves 80% efficiency

## Framing

- Break sequence of bits into a frame
- Typically implemented by network adaptor



## Approaches

CS 461

- Sentinel-based
  - delineate frame with special pattern: 01111110
  - e.g., HDLC, SDLC, PPP

8	16		16	8
Beginning sequence	Header	Body	CRC	Ending sequence

- problem: special pattern appears in the payload
- solution: bit stuffing
  - sender: insert 0 after five consecutive 1s
  - receiver: delete 0 that follows five consecutive 1s

## Approaches (cont)

- Counter-based
  - include payload length in header
  - e.g., DDCMP

8	8	8	14	42		16
SYN	SYN	Class	Count	Header	Body	CRC

- problem: count field corrupted
- solution: catch when CRC fails

Spring 2005

CS 461

## Approaches (cont)

- Clock-based
  - each frame is 125us long
  - e.g., SONET: Synchronous Optical Network
  - STS-*n* (STS-1 = 51.84 Mbps)



# Cyclic Redundancy Check

- Add *k* bits of redundant data to an *n*-bit message
  - want  $k \ll n$
  - 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<sup>3</sup> + x<sup>2</sup> + 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

Spring 2005

• Divide (P(x) + E(x)) by C(x); remainder zero if:

CS 461

- 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*<sup>*k*</sup> and *x*<sup>0</sup> 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 2.6 on page 102 for common C(x)

Spring 2005	CS 461	13	Spring 2005

## 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.

<pre>u short cksum(u_short *buf, int count) {     register u_long sum = 0;     while (count)</pre>	
<pre>{     sum += *buf++;     if (sum &amp; 0xFFFF0000)     {         /* carry occurred, so wrap around */         sum &amp;= 0xFFFF;         sum++;     } }</pre>	,
<pre>return ~(sum &amp; 0xFFFF); }</pre>	
2005 CS 461	

### Acknowledgements & Timeouts









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

Spring 2005

## 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 <= SWS



- Advance LAR when ACK arrives
- Buffer up to sws frames

Spring 2005

CS 461

18

## SW: Receiver

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



- Frame **SeqNum** arrives:
  - if LFR < SeqNum < = LFA  $\rightarrow$  accept
  - if SeqNum < = LFR or SeqNum > LFA  $\longrightarrow$  discarded
- Send cumulative ACKs

Spring 2005

19

17

## Sequence Number Space

- SeqNum field is finite; sequence numbers wrap around
- Sequence number space must be larger then 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 retransmits 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

Spring 2005

## **Concurrent Logical Channels**

- Multiplex 8 logical channels over a single link
- Run stop-and-wait on each logical channel
- Maintain three state bits per channel
  - channel busy
  - current sequence number out
  - next sequence number in
- Header: 3-bit channel num, 1-bit sequence num

CS 461

- 4-bits total
- same as sliding window protocol
- Separates *reliability* from *order*

Spring 2005