# Outline

- Chapter 2: Direct Link Networks

- Encoding
- Framing
- Error Detection
- Sliding Window Algorithm

# Direct Link Networks

# Direct Link Networks

- Hosts are directly connected by some medium
  - Twisted pair: telephone cable, Ethernet (Category 5: Cat5)
  - Coaxial pair: TV
  - Optical Fiber
  - Wireless: Infrared, Radio, Microwave

- Common bandwidth designators:
  - DS1 (or T1): 1.544 Mbps
  - DS3 (or T3): 44.736 Mbps (for example, Charter Athens has 2 DS3 links now)
  - STS-1 (OC1): 51.840 Mbps
  - STS-12: 622.080 Mbps …

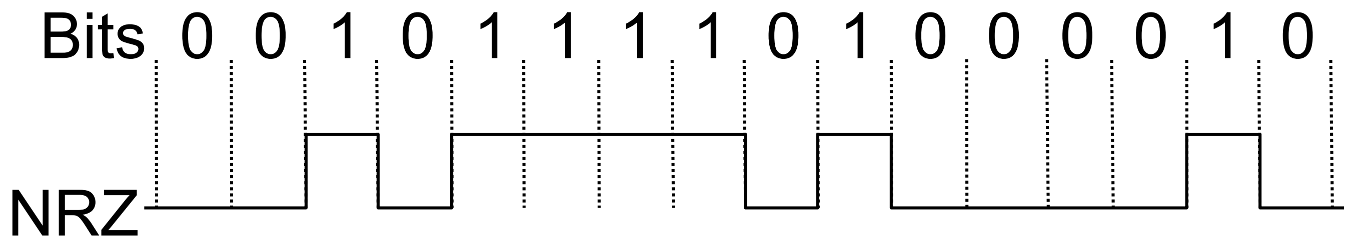
# Last Mile

- Plain Old Telephone Service) POTS:
  - 28.8 Kbps to 56 Kbps
- ISDN
- xDSL 1.544 Mbps to 8.448 Mbps
- Cable (40 Mbps down, 20 Mbps up) – Shared
  - wish we can get that much huh?

# Encoding

# 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)



# 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

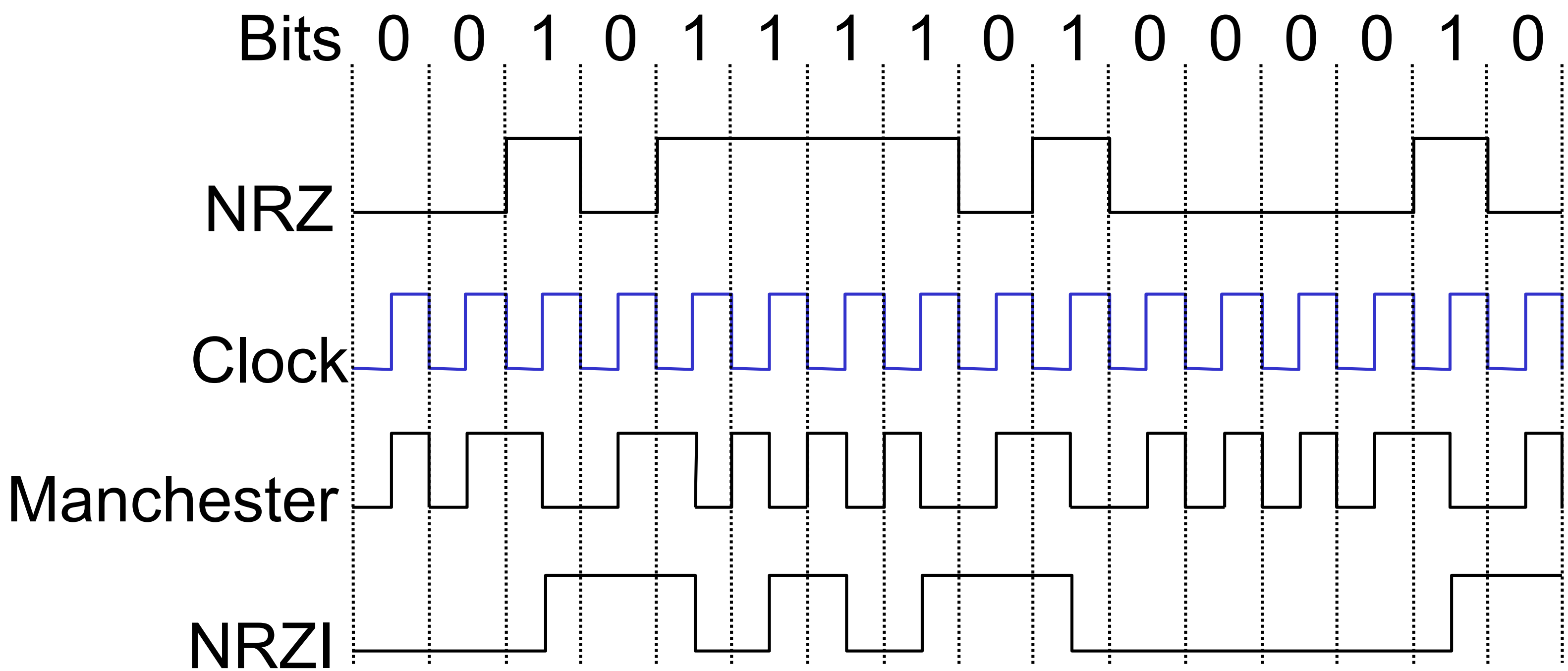
# Alternative Encodings

- 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.

# Encodings (cont)

- 4B/5B
  - 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

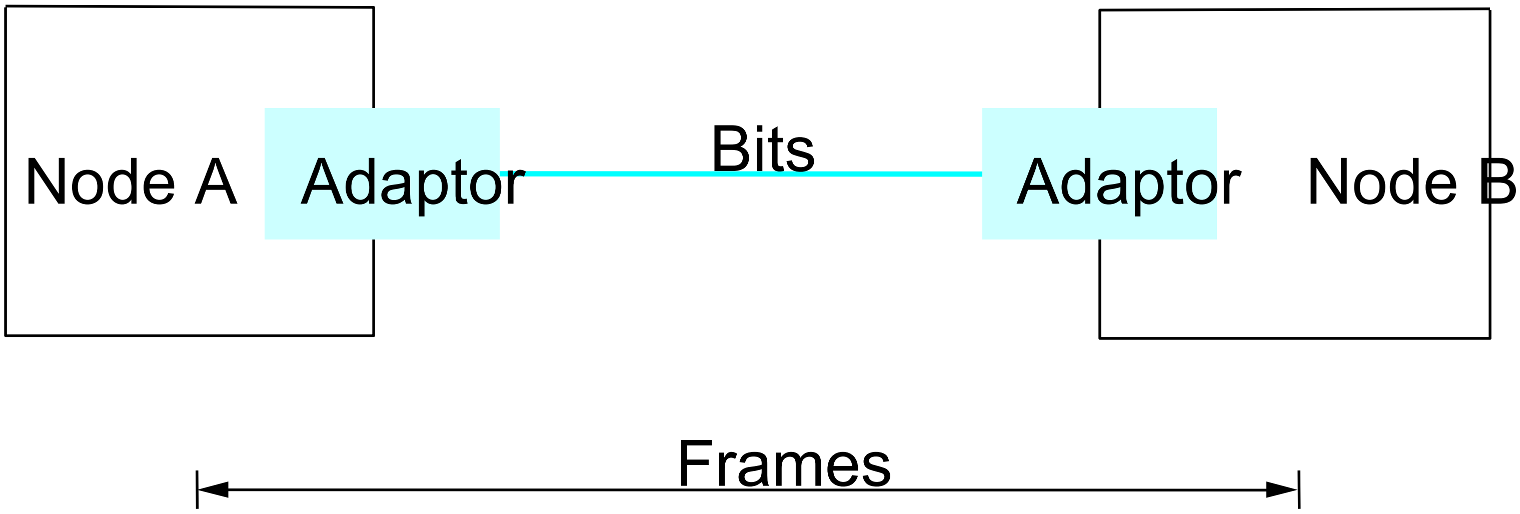
# Encodings (cont)



# Framing

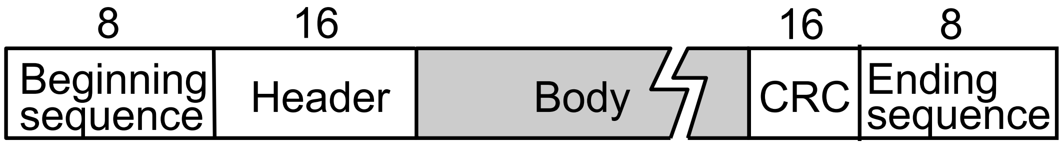
# Framing

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



# Approaches

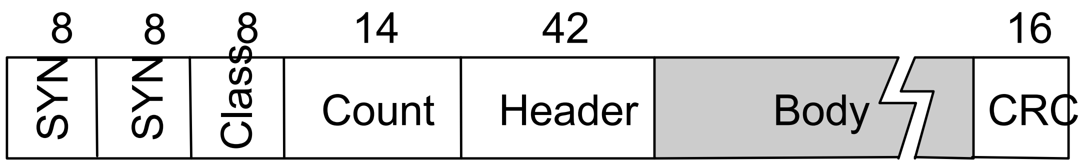
- Sentinel-based
  - delineate frame with special pattern: 01111110
  - e.g., HDLC (ISO), SDLC (IBM), PPP (dialup)




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

# Approaches (cont)

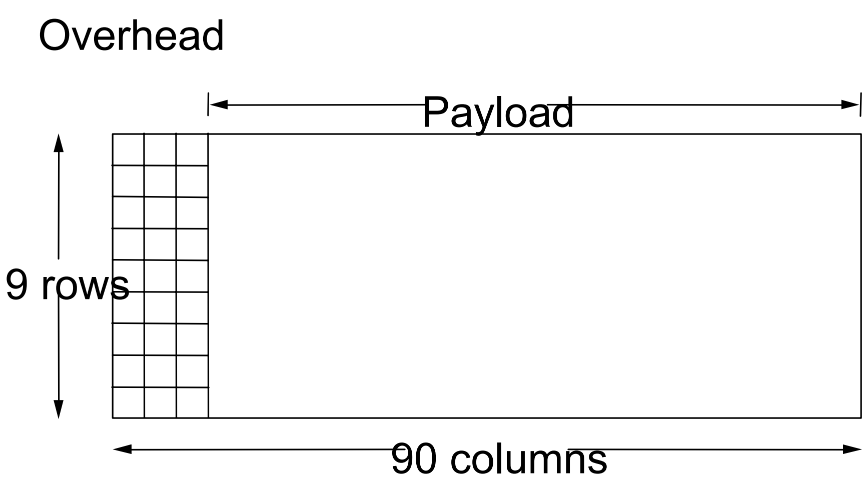
- Counter-based
  - include payload length in header
  - e.g., DDCMP (DECNET)

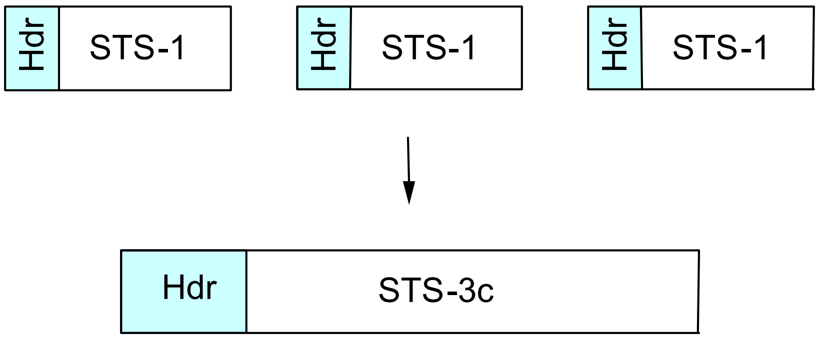



  - problem: count field itself corrupted
  - solution: catch when CRC fails

# Approaches (cont)

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




Three STS-1 frames multiplexed onto one STS-3c

# Error Detection

# 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) = x7 + x4 + x3 + x1
- Let k be the degree of some divisor polynomial
  - e.g., C(x) = x3 + x2 + 1

# CRC (cont)

- Transmit polynomial P(x) that is evenly divisible by C(x)
  - shift left k bits, i.e., M(x)xk
  - subtract remainder of M(x)xk / C(x) from M(x)xk
- 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 xk and x0 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)

# 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.

```
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);
}
```