Last time: Physical link layer

- Tasks
  - Encode binary data from source node into signals that physical links carry
  - Signal is decoded back into binary data at receiving node
  - Work performed by network adapter at sender and receiver
- Synchronous encoding algorithms
  - NRZ, NRZI, Manchester, 4B/5B

Encoding algorithms

- NRZ
  - High = 1
  - Low = 0

- NRZI
  - Transition = 1
  - Maintain = 0

- Manchester
  - High to low = 1
  - Low to high = 0

- 4B/5B
  - Use pre-selected 5 bits (with limited #s of zeros) to encode 4 bit sequences
  - Use NRZI to send signal
Today: Data-link layer

- Framing (2.3)
- Error detection/correction (2.4)

Framing

- In packet-switched networks, blocks of data (called frames), not bit streams, are exchanged between nodes
- Goal: Separate bit stream into frames (distinct units of transfer)
- Why?
  - Synchronization recovery
  - Link multiplexing
  - Efficient error detection
- Challenges
  - How can we determine exactly what set of bits constitute a frame?
  - How do we determine the beginning and end of a frame?

Framing

- Approaches
  - Sentinel (like C strings)
  - Length-based (like Pascal, SQLite)
  - Clock based
- Characteristics
  - Bit- or byte-oriented
  - Fixed or variable length
  - Data-dependent or data-independent length
Sentinel-based framing

- Basic idea: Identify start/end of frame with special “marker”
  - Byte pattern, bit pattern, signal pattern

- Challenge: What if marker is in data stream?

- Solution: “Character stuffing” used to recode data and insert extra escape characters
  - Analogous to escaping quotation marks with backslashes in strings in C

Byte-oriented sentinels

- STX – start of text
- ETX – end of text
- Problem: What if ETX appears in the data portion of the frame?
- Solution
  - If ETX appears in the data, introduce a special character DLE (Data Link Escape) before it – this is character/byte stuffing
  - If DLE appears in the text, introduce another DLE character before it
  - Efficiency can be only 50% (worst case)
- Protocol examples
  - BISYNC, PPP

Consistent-Overhead Byte Stuffing (COBS)

- Sentinel based framing
- Run length encoding applied to byte stuffing
  - Add implied 0 to end of frame
  - Each 0 is replaced with (number of bytes to next 0) + 1
  - What if no 0 within 255 bytes? – 255 value indicates 254 bytes followed by no zero
  - Worst case – no 0’s in packet – 1/254 overhead
- Appropriate for very low-bandwidth links

<table>
<thead>
<tr>
<th>Code</th>
<th>Followed by</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>(not applicable)</td>
<td>(not allowed)</td>
</tr>
<tr>
<td>0x01</td>
<td>No data bytes</td>
<td>A single zero byte</td>
</tr>
<tr>
<td>n</td>
<td>(n-1) data bytes</td>
<td>Data followed by 0</td>
</tr>
<tr>
<td>0xFF</td>
<td>254 data bytes</td>
<td>Data, no following 0</td>
</tr>
</tbody>
</table>

Code Followed by Meaning
0x00 (not applicable) (not allowed)
0x01 No data bytes A single zero byte
n (n-1) data bytes Data followed by 0
0xFF 254 data bytes Data, no following 0
How COBS works

Length-based framing
- Byte-counting approach
- End of frame
  - Calculated from length sent at start of frame
  - Challenge: Corrupt length markers
- Examples
  - DECNET’s DDCMP:
    - Byte-oriented, variable-length

Bit-oriented sentinel framing
- Not concerned with byte boundaries
  - Frame = collection of bits (variable length)
- Examples: HDLC and SDLC
- HDLC denotes beginning and end of frame with distinguished bit sequence 01111110
- Use analog of character stuffing to escape “01111110” called bit stuffing
- Sender stuffs 0 after any string of five 1’s (except for 01111110)
  - 011111 followed by 0 → bit stuffing
  - 011111 followed by 10 → end of frame marker
  - 011111 followed by 11 → error
Clock-based framing

- Continuous stream of fixed-length frames
- Clocks must remain synchronized
  - No bit or byte stuffing

Example:
- Synchronous Optical Network (SONET)

Problems:
- Frame synchronization – when do frames start and end
- Clock synchronization – sender and receiver must have synchronized clocks

SONET

- All frames (STS formats) are 125 μsec long
- Problem: How do we recover frame synchronization
  - 2-byte synchronization pattern starts each frame (unlikely in data)
  - Wait until pattern appears in same place repeatedly
- Problem: How do we maintain clock synchronization
  - NRZ encoding, data scrambled (XOR'd) with 127-bit pattern
  - Creates transitions to avoid longs strings of 0s and 1s in NRZ
  - Also reduces chance of finding false sync. pattern

Error Detection

- Bit errors are sometimes introduced in frames
- Goal: Validate “correctness” of frame
- Idea: Send additional redundant (add no new information) data with frame to check if it has been damaged

- Checked at many layers
  - Physical (e.g. modulation)
  - Datalink (e.g. cyclic redundancy check)
  - Network/Transport (e.g. IP Checksum)
  - Application (e.g. MD5 hash)

- Today: simple parity, redundancy w/voting, 2-dimensional parity, IP checksum, CRCs
Error Detection from 10,000 feet

- EDC = Error Detection bits (redundancy)
- D = Data protected by error checking, may include header fields
- Error detection not 100% reliable!
  - Protocol may miss some errors, but rarely
  - Larger EDC field yields better detection

Simple parity

- 1-bit error detection with parity
  - Add an extra bit to a code to ensure an even (odd) number of 1s
  - Every code word has an even (odd) number of 1s
  - If # of 1s = odd, add 1 as parity bit
  - If # of 1s = even, add 0 as parity bit

Voting

- 1-bit error correction with voting
  - Every codeword is transmitted n times
  - "Majority rules"
  - Doesn't always work
Hamming distance

- The Hamming distance between two code words is the minimum number of bit flips to move from one to the other
  - Example: 00101 and 00010
    Hamming distance of 3
- The minimum Hamming distance of a code is the minimum distance over all pairs of codewords
  - Minimum Hamming Distance for parity = 2
  - Minimum Hamming Distance for voting = 3
- N-bit error detection
  - No code word changed into another code word
  - Requires Minimum Hamming Distance of N+1

Two-dimensional parity

- Use 1-dimensional (simple) parity
  - Add one bit to a 7-bit code to ensure an even/odd number of 1s
- Add 2nd dimension
  - Add an extra byte to frame
    - Bits are set to ensure even/odd number of 1s in that position across all bytes in frame
- Comments
  - Catches all 1-, 2- and 3-bit and most 4-bit errors

Internet checksum

- Idea
  - Add up all the words
  - Transmit the sum called the checksum
  - Receiver performs same calculation
- Internet Checksum
  - Assume data is sequence of 16-bit 1’s complement integers
  - Find 1’s complement sum of sequence of 16-bit codewords
  - Take 1’s complement of resulting sum = checksum
  - Example
    - Message: e34f 2396 4427 99f3
      - 2s complement sum is 1e4ff
      - 1s complement sum is e4ff + 1 = e500
      - Checksum is 1aff
- Comments
  - VERY easy to implement, fast incremental updates
  - Not very robust
IP Checksum

```c
u_short checksum(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 (CRC)

- **Polynomial code**
  - Treat packet bits a coefficients of n-bit polynomial
    - Message = 10011010
    - Generator polynomial
      \[ = x^7 + x^4 + x^3 + x^{\text{r+1}} \]
  - Choose r+1 bit generator polynomial (well known – chosen in advance)
  - Add r bits to packet such that message is divisible by generator polynomial
  - Note: easy way to think of polynomial arithmetic mod 2
    - Multiplication: binary addition without carries
    - Division: binary subtraction without carries
- Better loss detection properties than checksums

Error Detection – CRC

- View data bits, D, as a binary number
- Choose r+1 bit pattern (generator), G
- Goal: choose r CRC bits, R, such that
  - \(<D,R>\) exactly divisible by G (modulo 2)
  - Receiver knows G, divides \(<D,R>\) by G. If non-zero remainder: error detected!
  - Can detect all burst errors less than r+1 bits
- Widely used in practice (Ethernet, FDDI, ATM)
Common Generator Polynomials

<table>
<thead>
<tr>
<th>CRC</th>
<th>Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC-8</td>
<td>(x^8 + x^7 + x^4 + 1)</td>
</tr>
<tr>
<td>CRC-10</td>
<td>(x^{10} + x^9 + x^8 + x^4 + x + 1)</td>
</tr>
<tr>
<td>CRC-12</td>
<td>(x^{12} + x^{11} + x^6 + x^5 + x + 1)</td>
</tr>
<tr>
<td>CRC-16</td>
<td>(x^{16} + x^{15} + x^2 + x + 1)</td>
</tr>
<tr>
<td>CRC-CCITT</td>
<td>(x^{16} + x^{12} + x^5 + 1)</td>
</tr>
<tr>
<td>CRC-32</td>
<td>(x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1)</td>
</tr>
</tbody>
</table>

CRC – Example Encoding

\[\begin{align*}
x^3 + x^2 + 1 & = 1101 \quad \text{Generator } g \text{ (degree } r) \\
x^3 + x^2 + x^4 + x & = 10011010000 \quad \text{Message } m \\
10011010000 & \quad \text{Message plus } r \text{ zeros} \\
\overbrace{\ldots} & \quad \text{Result:} \\
\text{Transmit message} \quad & \quad \text{followed by} \\
\text{remainder:} & \quad 10011010101 \\
\end{align*}\]

CRC – Example Decoding – No Errors

\[\begin{align*}
x^3 + x^2 + 1 & = 1101 \quad \text{Generator } g \\
x^{16} + x^2 + x^4 + x^4 + x^2 + 1 & = 10011010101 \quad \text{Received Message } m \\
\overbrace{\ldots} & \quad \text{Result:} \\
\text{Received message, no} \quad & \quad \text{errors} \\
\text{CRC test is passed} & \quad \text{CRC test is passed} \\
\end{align*}\]
CRC – Example Decoding – with Errors

\[ x^3 + x^2 + 1 \]
\[ x^4 + x^3 + x^2 + x + 1 \]
\[ r + 1 \text{ bit check sequence } g, \text{ equivalent to a degree-}r \text{ polynomial} \]

Generator \( g \) = 1001010101
Received Message \( m \) = 100101101

Remainder \( m \mod g \)

Two bit errors
Result:
CRC test failed

Summary

- Framing
  - Bunching bits into distinct messages (frames)
  - Challenge is in finding where one frame starts and another begins

- Error detection
  - Determine if frame is corrupted by checking it against redundant data

- Next time: more on the datalink layer
  - Reliable Transmission: Section 2.5

Administrative details

- Project 1 is out – due next next Monday (7/11)
- No discussion next week – 4th of July Holiday

- Office Hours for project help are \textit{VERY} important
- Neha and I alternate office hours – one every day of the class week!
- We have a “make up session” scheduled for Fri, July 8
  - Can use that for project discussion / extended OH/prj help

- Please see me after class if you cannot access ieng6.ucsd.edu