Outline

- Encoding and decoding
  - Translate between bits and digital signal
- Framing
  - Bit stream to packets
- Packet loss & corruption
  - Error detection
  - Flow control
  - Loss recovery

From Signals to Packets

Link Layer: Implementation

- Implemented in "adapter"
  - E.g., Ethernet card or chip
  - Typically includes: RAM, DSP chips, host bus interface, and link interface
Datalink Functions

- Encoding: change bit stream before transmission
- Framing: encapsulating a network layer datagram into a bit stream.
  - Add header, mark and detect frame boundaries
- Error control: error detection and correction to deal with bit errors.
  - May also include other reliability support, e.g. retransmission
- Flow control: avoid that sender outruns the receiver
- Media access: controlling which frame should be sent next over datalink.
- Hubbing, bridging: extend the size of the network

Why Do We Need Encoding?

- Keep receiver synchronized with sender.
- Create control symbols, in addition to regular data symbols.
  - E.g. start or end of frame, escape, ...
- Error detection or error corrections.
  - Some codes are illegal so receiver can detect certain classes of errors
  - Minor errors can be corrected by having multiple adjacent signals mapped to the same data symbol
- Encoding can be done one bit at a time or in multi-bit blocks, e.g., 4 or 8 bits.
- Encoding can be very complex, e.g. wireless

How to Encode?

- Seems obvious, why waste time on this? Just modulate the signal!

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
\hline
.85 & 0 & & & & & & & .85 \\
\hline
.85 & 0 & & & & & & & .85 \\
\hline
.85 & 0 & & & & & & & .85 \\
\hline
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
\end{array}
\]

- But:
  - How easily can the receiver retrieve the bit stream?
  - What happens when there are errors: a bit gets flipped?

How about the Poor Receiver?

- Sender needs to help the receiver by “shaping” the digital bit stream so it easy to correctly interpret
  - Applies to combination of modulation and coding
- Problem in this case: not enough transitions
Simple Encoding Examples

- Change the bit stream so there are enough transitions in the signal
  - Helps the receiver
  - But this also increases the complexity of the signal
  - Uses more spectrum!
- Manchester encoding was used in the original ethernet

Manchester Encoding

- Used by Ethernet
- 0=low to high transition, 1=high to low transition.
- Transitions simplify clock recovery and good electrical properties for any bit stream
- But you pay a price!
  - Doubles the number of transitions – more spectrum!
  - Circuitry must run twice as fast

Take-away: Encoding and Modulation

- Encoding and modulation work together
  - Must generate a signal that works well for the receiver – has good electrical properties
  - Must be efficient with respect to spectrum use
  - Can shift some of the burden between the two layers
  - Tradeoff is figured out by our electrical engineers
- Maintaining good electrical properties
  - Spectrum efficient modulation requires more encoding
  - For example: 4B/5B encoding (later)
- Error recovery
  - Aggressive modulation needs stronger coding

Outline

- Encoding and decoding
  - Translate between bits and digital signal
- Framing
  - Bit stream to packets
- Packet loss & corruption
  - Error detection
  - Flow control
  - Loss recovery
Framing

- A link layer function, defining which bits have which function.
- Minimal functionality: mark the beginning and end of packets (or frames).
- Some techniques:
  - Out of band delimiters (e.g. 4B/5B control symbols)
  - Frame delimiter characters with character stuffing
  - Frame delimiter codes with bit stuffing
  - Synchronous transmission (e.g. SONET)
    - Boundaries are based on timing

Delimiter Based

- SYN: sync character
- SOH: start of header
- STX: start of text
- ETX: end of text
- What happens when ETX is in Body?

Character and Bit Stuffing

- Mark frames with special character.
  - What happens when the user sends this character?
  - Use escape character when a control symbol appears in data:
    - *abc*def \rightarrow *abc\*def
    - Very common on serial lines, in editors, etc.
  - Mark frames with special bit sequence (read)
    - must ensure data containing this sequence can be transmitted
    - example: suppose 11111111 is a special sequence.
    - transmitter inserts a 0 when this appears in the data:
      - 11111111 \rightarrow 111111101 – receiver deletes a 0 after seven 1’s
      - Means that we must stuff a zero any time seven 1s appear:
        - 11111110 \rightarrow 111111100
        - receiver unstuffs: 111111100 \rightarrow 11111110
Ethernet Framing

- Preamble is 7 bytes of 10101010 (5 MHz square wave) followed by one byte of 10101011
- Allows receivers to recognize start of transmission after idle channel

4B/5B Encoding

- Data coded as symbols of 5 line bits → 4 data bits, so 100 Mbps uses 125 MHz.
- Encoding ensures no more than 3 consecutive 0’s
- Uses NRZI to encode resulting sequence
  - Uses less frequency than Manchester encoding
  - 16 data symbols, 8 control symbols
  - Data symbols: 4 data bits
  - Control symbols: idle, begin frame, etc.
- Example: FDDI.

4B/5B Encoding

<table>
<thead>
<tr>
<th>Data</th>
<th>Code</th>
<th>Data</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>11110</td>
<td>1000</td>
<td>10010</td>
</tr>
<tr>
<td>0001</td>
<td>01001</td>
<td>1001</td>
<td>10011</td>
</tr>
<tr>
<td>0010</td>
<td>10100</td>
<td>1010</td>
<td>10110</td>
</tr>
<tr>
<td>0011</td>
<td>10101</td>
<td>1011</td>
<td>10111</td>
</tr>
<tr>
<td>0100</td>
<td>01010</td>
<td>1100</td>
<td>11010</td>
</tr>
<tr>
<td>0101</td>
<td>01011</td>
<td>1101</td>
<td>11011</td>
</tr>
<tr>
<td>0110</td>
<td>01110</td>
<td>1110</td>
<td>11100</td>
</tr>
<tr>
<td>0111</td>
<td>01111</td>
<td>1111</td>
<td>11101</td>
</tr>
</tbody>
</table>

Other Encodings

- 8B/10B: Fiber Channel and Gigabit Ethernet
- 64B/66B: 10 Gbit Ethernet (& 40 and 100 Gb/S)
- B8ZS: T1 signaling (old - bit stuffing)

Things to Remember

- Encoding and modulation must work together
- Lots of approaches
- Rule of thumb:
  - Little bandwidth → complex encoding
  - Lots of bandwidth → simple encoding
Outline

- Encoding
  - Bits to digital signal
- Framing
  - Bit stream to packets
- Packet loss & corruption
  - Error detection
  - Flow control
  - Loss recovery

Error Coding

- Transmission may introduce errors into a message.
  - Received “digital signal” is different from that transmitted
  - Single bit errors versus burst errors
- Detection:
  - Requires a convention that some messages are invalid
  - Hence requires extra bits
  - An (n,k) code has codewords of n bits with k data bits and r = (n-k) redundant check bits
- Correction
  - Forward error correction: many related code words map to the same data word
  - Detect errors and retry transmission

Error Detection

- EDC= Error Detection and Correction 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 and correction

Parity Checking

Single Bit Parity:
Detect single bit errors

\[
\begin{align*}
01110001101010110
\end{align*}
\]
Internet Checksum

- Goal: detect “errors” (e.g., flipped bits) in transmitted segment

**Sender**
- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1’s complement sum) of segment contents
- Sender puts checksum value into checksum field in header

**Receiver**
- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. But maybe errors nonetheless?

Basic Concept: Hamming Distance

- Hamming distance of two bit strings = number of bit positions in which they differ.
  
  | 1 0 1 0  | HD=2  
  | 1 1 0 1 0 |

- If the valid words of a code have minimum Hamming distance D, then D-1 bit errors can be detected.
- If the valid words of a code have minimum Hamming distance D, then [(D-1)/2] bit errors can be corrected.

Cyclic Redundancy Codes (CRC)

- Commonly used codes that have good error detection properties.
  - Can catch many error combinations with a small number of redundant bits
- Based on division of polynomials.
  - Errors can be viewed as adding terms to the polynomial
  - Should be unlikely that the division will still work
- Can be implemented very efficiently in hardware.
- Examples:
  - CRC-32: Ethernet
  - CRC-8, CRC-10, CRC-32: ATM

CRC: Basic idea

- Treat bit strings as polynomials:

  \[ X^3 + X^2 + X^1 + X^0 \]

- Sender and Receiver agree on a divisor polynomial of degree k
- Message of M bits → send M+k bits
- No errors if M+k is divisible by divisor polynomial
- If you pick the right divisor you can:
  - Detect all 1 & 2-bit errors
  - Any odd number of errors
  - All Burst errors of less than k bits
  - Some burst errors ≥ k bits
Outline

- Encoding
  - Bits to digital signal
- Framing
  - Bit stream to packets
- Packet loss & corruption
  - Error detection
  - Flow control
  - Loss recovery

What is Used in Practice?

- No flow or error control.
  - E.g. regular Ethernet, just uses CRC for error detection
- Flow control only
  - E.g. Gigabit Ethernet
- Flow and error control.
  - E.g. X.25 (older connection-based service at 64 Kbs that guarantees reliable in order delivery of data)
  - Flow and error control solutions also used in higher layer protocols
    - E.g., TCP for end-to-end flow and error control