Last Time/Today

- Project on class website – short overview today
- Reading set with critique due next Monday

- Last Time
  - Finished Physical Layer
  - End to End principle paper (Saltzer 81)
  - Design considerations for the Internet (Clark 88)

- Today:
  - Link layer
Communication using a direct link – Issues

- Encoding: should we just pass the 1’s and 0’s as is?
- Framing: where does the data begin and where does it end?
- Error Detection (and, possibly, correction)
- Medium Access Control (if not point to point)
Communication using a direct link – Issues

- Encoding: should we just pass the 1’s and 0’s as is?
- Framing: where does the data begin and where does it end?
- Error Detection (and, possibly, correction)
- Medium Access Control (if not point to point)
Non-Return to Zero (NRZ)

- NRZ: data bits are transmitted as is
- Problem: Consecutive 1’s or 0’s
  - Baseline wander:
    * average receive power is used as the base line after demodulation
    * when a new bit is received, it is compared to the baseline to know if 1 or 0
  - Cannot recover clock – clock drift
- Low signal, or no signal?
Better Encoding – NRZI and Manchester

- Non Return to Zero Inverted (NRZI) – transition value of signal to indicate a 1, otherwise keep it the same
  - Why is this useful?

- Manchester Encoding: Transmit XOR of NRZ and the clock
  - Solves all our problems – but good enough?
4B/5B Encoding

<table>
<thead>
<tr>
<th>4-bit Data Symbol</th>
<th>5-bit Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>11110</td>
</tr>
<tr>
<td>0001</td>
<td>01001</td>
</tr>
<tr>
<td>0010</td>
<td>10100</td>
</tr>
<tr>
<td>0011</td>
<td>10101</td>
</tr>
<tr>
<td>0100</td>
<td>01010</td>
</tr>
<tr>
<td>0101</td>
<td>01011</td>
</tr>
<tr>
<td>0110</td>
<td>01110</td>
</tr>
<tr>
<td>0111</td>
<td>01111</td>
</tr>
<tr>
<td>1000</td>
<td>10010</td>
</tr>
<tr>
<td>1001</td>
<td>10011</td>
</tr>
<tr>
<td>1010</td>
<td>10110</td>
</tr>
<tr>
<td>1011</td>
<td>10111</td>
</tr>
<tr>
<td>1100</td>
<td>11010</td>
</tr>
<tr>
<td>1101</td>
<td>11011</td>
</tr>
<tr>
<td>1110</td>
<td>11100</td>
</tr>
<tr>
<td>1111</td>
<td>11101</td>
</tr>
</tbody>
</table>

- Idea: Every 4 bits encoded as 5-bit code such that
  - Each code has no more than 1 leading 0 or 2 trailing 0’s
  - Will never have more than 3 0’s in a row
  - Transmit using NRZI – consecutive 1’s not a problem
  - 80% efficiency!!
  - Can we utilize the 16 unused symbols?
Second Problem: Framing

- Blocks of data are exchanged across the links – frames
  - NIC fetches/deposits frames out of/into node memory

- Framing:
  - How do we tell where a frame begins and where it ends?
  - Pretty straightforward – any ideas?

- How is this implemented (hardware or software?); what layer does it conceptually belong to?
Framing Alternatives

- Sentinels: use start and end markers
  - must make sure they don't appear in payload – bit/byte stuffing

- Use count:
  - Still need a start marker
  - Is stuffing necessary?

- clock-based:
  - Use a constant size payload – operation synchronous
  - Need to be able to find the start of the frame still
Clock-Based Framing (SONET)

- Idea: use fixed frame size, do not need to supply count

- STS-1 frame shown above; 9 rows of 90 bytes each, 3 of which are “header”

- First two bytes are a special start of frame pattern

- Scrambled NRZ used (XOR’d with a special bit pattern)

- What if header appears in payload?
Problem 3 – Error Detection and Correction

- Transmission media are susceptible to transmission errors
  - Single bit errors vs. burst errors
- Minimal capability – error detection
- For reliable transmission need error correction or detection/retransmission
- Aim of Error detection – detect as many errors as possible using as little overhead as possible
- Example: send 1 parity bit (parity is bit-wise XOR)
Better algorithm – 2D Parity

- Capable of detecting all 1, 2 and 3 bit errors as well as most 4-bit errors (not all? give example)
- for an X by Y message, X + Y + 1 bits overhead
- Another algorithm: Internet checksum – unsigned addition of the message words
Internet Checksum Algorithm

u_short cksum(u_short *buf, int count) {
    register u_long sum = 0;
    while(count --) {
        sum += *buff++;
        if (sum & 0xffff0000) {
            /* carry occurred; wrap around*/
            sum &=0xffff; sum++; }
    }
    return ~((sum & 0xffff));
}

- Add up all the message words (16-bit) to produce a 1’s complement sum; send the result with the message

- Not used at the link layer

- Overhead is small (16-bits), but how good is it?
Cyclic Redundancy Check (CRC)

- Uses Polynomial Modulo 2 arithmetic to provide strong, low-overhead, error-detection

- An n-bit message represented as an n-1 degree polynomial
  - Example: MSG = 10011010 corresponds to \( M(x) = x^7 + x^4 + x^3 + x^1 \)

- Each scheme has a specified divisor polynomial, chosen to detect the most errors
  - Example: \( C(x) = x^3 + x^2 + 1 \)

- Idea: Add k-bits of redundant data to an n-bit message such that the resultant “polynomial” is perfectly divisible by \( C(x) \) (i.e., remainder = 0)
How to find the k-bits

- Sender Multiplies $M(x)$ by $x^k$ (that is, the message is first padded with $k$ zeros)
- Divide the result by $C(x)$ – Rules
  - polynomials of the same degree are divisible
  - subtraction is a bit-wise XOR

\[
\begin{array}{c}
\text{Generator} \rightarrow 1101 \\
\downarrow \quad \downarrow \\
11111001 \\
\downarrow 100110000 \\
\downarrow 1101 \\
1001 \\
\downarrow 1101 \\
1000 \\
\downarrow 1101 \\
1011 \\
\downarrow 1101 \\
1100 \\
\downarrow 1101 \\
1000 \\
\downarrow 1101 \\
101 \quad \text{Reminder}
\end{array}
\]

- Subtract (XOR) the remainder from padded message
- Choose $C(x)$ such that errors very rarely result in divisible messages; Why/How does this work?
Why this Works

- $C(x)$ can be chosen such that the following errors are detectable
  - 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 $C(x)$ has a factor of at least 3 terms (is perfectly divisible by it)
  - Any odd number of errors as long as $C(x)$ has the factor $(x + 1)$
  - Any burst error for which the length of the burst is less the $k$-bits

- Many other errors are also detectable (but not all)

- Try to prove some of these properties
CRC (cont’d)

<table>
<thead>
<tr>
<th>CRC</th>
<th>C(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC-8</td>
<td>$x^8 + x^2 + x + 1$</td>
</tr>
<tr>
<td>CRC-10</td>
<td>$x^{10} + x^9 + x^5 + x^4 + x + 1$</td>
</tr>
<tr>
<td>CRC-12</td>
<td>$x^{12} + x^{11} + x^3 + x^2 + 1$</td>
</tr>
<tr>
<td>CRC-16</td>
<td>$x^{16} + x^{15} + x^2 + 1$</td>
</tr>
<tr>
<td>CRC-CCITT</td>
<td>$x^{16} + x^{12} + x^5 + 1$</td>
</tr>
<tr>
<td>CRC-32</td>
<td>$x^{32} + \ldots + x + 1$</td>
</tr>
</tbody>
</table>

- The 6 polynomials above are in use
  - Note that CRC-X uses an x+1 bit C(x), but produces an x-bit remainder

- Efficiently Generated in hardware using shift registers and XOR gates

- What happens after an error is detected?
Another Shannon Result

“Communication can be done over an error prone medium as long as the data rate is less than or equal the channel capacity with a vanishingly small error probability”

• i.e., errors can be corrected!
  – Forward Error Correction/Error Correcting Codes (include enough redundancy to correct errors)
  – Retransmission

• Which should be used?
Tradeoff

• FEC/ECC
  – FEC provides constant throughput and predictable delay
  – If high error rate, need long codes/complex circuitry
  – Does not protect against all errors, or packet loss

• Retransmission
  – Unpredictable delay
  – Have to have a feedback channel
Forward Error Correction (or Error Correcting Codes)

- General idea is to include enough redundant information to allow recovery from many errors

- Majority Encoding: Repeat every message (or character) several times; assume the majority is correct

- Example: Transmit HHEEELLLLLLOOO WWOOORRRLLLLDDDD instead of HELLO WORLD
  - Can recover from 1 character errors
  - If you receive HAHEEELL1aLLO!O WWWOO)RRLLL1DD you can recover the message by taking majority for each character
Simple Analysis of This Scheme

- Suppose the probability of error on any symbol is 0.05
  - With no redundancy, probability of correct reception is $(0.95)^{10} = 60\%$
  - With redundancy
    * the probability of receiving a symbol correctly is:
      $(0.95)^3 + 3 (0.95)^2 (0.05) = 0.99275$
      * The probability of correct reception is $(0.99275)^{10} = 93\%$

- Great improvement in reliability, but at the cost of tripling our transmitted symbols!

- Must be able to do better
- Let the message blocks and code words be as follows (for Hamming code 4, 7)

\[
\begin{align*}
    m &= (m_0, m_1, m_2, m_3) \\
    c &= (m_0, m_1, m_2, m_3, b_0, b_1, b_2) \\
    b_0 &= (m_1 + m_2 + m_3) \\
    b_1 &= (m_0 + m_1 + m_3) \\
    b_2 &= (m_0 + m_2 + m_3)
\end{align*}
\]
A little Intuition – *Distance*

- How do spell checkers work?
  - Suppose you receive LIND, what are the possible original transmissions?
  - English Words are “close” to each other, difficult to correct sometimes
  - Can we find a language where blocks are not close?

- On the Other hand ...
  “rhearecsers in Cmdrabgie hvae funod out taht the haumn mnid deos not raed ervey lteter. So it can fgieru out waht is witretn as Inog as the frsit and Isat Irttees are crorcet. The rset can be a mses!”
Hamming Distance

- The Hamming Distance between two words \( x \) and \( y \) (\( H(x,y) \)) is the number of places where they differ
  - What is the Hamming Distance between 001011 and 101001?
  - \( H(x,x) = 0 \)
  - \( H(x,y) = H(y,x) \)
  - \( H(x,y) + H(y,z) \geq H(x,z) \) – can you prove it?

- The minimum distance across between words in a code determines the error tolerance of the code

- Many schemes have been proposed (Linear codes, covering codes, cyclic codes, convolution codes...); optional paper to dig deeper
BCH/Reed-Solomon Codes

- Given any “k” symbols, add N-k symbols of error correcting codes
- Maximum errors that can be detected is floor((N-k)/2) – optimal!
- Select N to balance overhead and correction ability: usually \( \frac{k}{N} > 0.8 \)
- Example: suppose \( k \) is 186 bits long, how many errors can be detected if N is 200 bits long?
- BCH developed the binary codes; RS extended to symbols
- More next time when we discuss digital fountain
Reliable Transmission

• Every packet has to be received correctly
  – Need to recover from corrupt frames (bit errors)
  – What about missing frames? (lost packets)

• Bit Errors
  – Correct them using Error Correction Codes?

• What about missing packets?

• What about delayed packets?

• Automatic Repeat reQuest (ARQ)
  – Retransmit corrupt/missing frames
FEC vs. ARQ

- At the link layer, reasonable to assume errors due to corrupt frames
- Should we use FEC or ARQ?
  - What is the tradeoff?
How to implement a retransmission based reliability

• How about:
  – If packet is received without an error, send an ACK
  – If packet is received with an error, send a Negative Acknowledgement (NACK)
  – If you receive an NACK, retransmit the packet

• Why do we need to send an ACK?

• Problems?
  – What if the ACK or NACK is corrupted?
  – What if packets or acknowledgements get lost?
Automatic Repeat ReQuest (ARQ)

- Idea: use acknowledgements and timeouts

- What are the problems/drawbacks?
  - Potential of duplication of packets
  - How to discriminate packet from duplicate?
  - Delayed packets?