Sockets Tutorial – in class or outside (both?)

Last Time: Finished defining requirements for scalable heterogeneous network
- Functionality (reliability, in-order-delivery, duplicate suppression, security...)
- Performance (bandwidth vs. latency; average vs. variance)

Building a network that meets these requirements
- Layering and encapsulation
- OSI 7 layer
- Internet protocol (TCP/IP)

Quick tour of Digital communication (or the physical layer)

Today
- Start Directly Connected Networks
Performance

- Bandwidth: how much data can I send per second
- Latency: Propagation + Transmission + queueing
  - Propagation delay: how long it takes a bit to cross the wire
  - Transmission: how long it takes to send all the data given bandwidth
  - Queueing: delay within intermediate nodes

Which factors are more important for a small message? for a large message?
Physical Communication

- Different Mediums “pass” signals with different frequency ranges
  - Glass passes visible light frequency but concrete doesn’t
- Sinusoid is the most efficient signal from a frequency perspective (has only one frequency component)
Physical Communication (cont’d)

- Idea of Physical communication
  - Use a carrier sinusoidal frequency to put you in the range that your medium passes/is available to you
  - Encode information by *modulating* this signal

- Three main types of modulation: Amplitude, Frequency, and Phase modulation

- Shannon’s Limit
Network Architecture: Layering

- Implement a complex system as an ordered series of abstractions called layers

- Each layer provides a service that depends only on the previous, less abstract, layer

<table>
<thead>
<tr>
<th></th>
<th>Application programs</th>
<th>Process-to-process channels</th>
<th>Host-to-host connectivity</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request/reply channel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message stream channel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host-to-host connectivity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Can have multiple abstractions at each layer
Encapsulation and Demultiplexing keys

- A lower level protocol treats the full packet (header/body) as its payload – **encapsulation**
- Multiplexing/demultiplexing: have to include a demux key to identify application a packet is destined for
Internet Architecture (TCP/IP)

- "Hourglass" shaped – IP is a common protocol that connects everything
- Does not imply strict layering
- The Socket API provides the service interface exported by the Internet protocol to users
How to communicate using a direct link – Issues

- Encoding
- Framing
- Error Detection (and, possibly, correction)
- Medium Access Control (if not point to point)
Problems with Basic Encoding

<table>
<thead>
<tr>
<th>Bits</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- NRZ: data bits are transmitted as is – why not?
- Problem 1: the sender and receiver are not synchronized
  - When there are a number of consecutive 0’s or 1’s, cannot recover clock – *clock drift*
- Problem 2: sender/receiver do not use the same reference voltage levels (“ground” varies from one location to another)
  - Baseline: 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
  - More 1’s than 0’s (or vice versa) we get Baseline wander
- 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?

- How is this implemented (hardware or software?); what layer does it conceptually belong to?
Byte Oriented Protocols

- Sentinel Approach – BISYNC

<table>
<thead>
<tr>
<th>SYN</th>
<th>SYN</th>
<th>SOH</th>
<th>Header</th>
<th>STX</th>
<th>Body</th>
<th>ETX</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td></td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>

- What are some problems that can happen with this approach?
  - What if the sentinels appeared in the payload?
  - How big is the frame?
Point-to-Point Protocol (RFC 1661/1547; STD 51)

- Used on many point to point links, including modems
- Provides:
  - Basic Framing, configurable format
  - A Link Control protocol (LCP) used to establish, configure and test the connection
  - A family of Network Control Protocols to interface with the network protocol (IP, IPX, etc...)
- Character stuffing used
- 52 RFCs in the PPP extension group
Alternative – Supply Byte Count

• Supply the number of bytes in the header; adapter can figure out when the frame ends

• Example DECNET’s DDCMP

<table>
<thead>
<tr>
<th>SYN</th>
<th>SYN</th>
<th>Class</th>
<th>Count</th>
<th>Header</th>
<th>Body</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>14</td>
<td>42</td>
<td></td>
<td>16</td>
</tr>
</tbody>
</table>

• Is character stuffing still needed?

• What if the count field gets corrupted?

• Is a sentinel still needed at the start of the frame? What if header appears in payload?
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?
STS-N/STS-Nc

- Each frame is 125 $\mu$-sec long; STS-1 \( \frac{810 \times 8}{125 \times 10^{-6}} = 51.84 \) MBps
- An STS-N circuit is made up of N STS-1 circuits byte interleaved (the frame size is 810 bytes * N)
  - N interleaved STS-1 circuits
- Can have a single circuit take the whole payload
  - STS-Nc (concatenated)
- Payload may float (header will point to start of payload)
Problem 3 – Error Detection and Correction

- All transmission media are susceptible to transmission errors to varying degrees
  - Single bit errors vs. burst errors
- What can be done to ensure error free communication?
  - What is the minimal capability needed? Need to at least be able to tell when an error occurred
  - Can we also correct the error?
- Error detection algorithm: send each frame twice, and compare the two copies to each other
- Error detection algorithm: send a single parity bit with the frame
- Aim of Error detection – detect as many errors as possible using as little overhead as possible
A 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 are added

- If an error is “detected” in this scheme, is it also “correctable”? 
Internet Checksum Algorithm

```c
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

  Generator $\rightarrow$ 1101 $\overline{11111001}$ $\overline{10011011000}$ Message

  1101
  11111001
  1001
  1101
  1000
  1101
  1100
  1101
  101 Remainder

- 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 divisiable 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
- Efficiently Generated in hardware using shift registers and XOR gates
- What happens after an error is detected?
Reliable Transmission

- Need to recover from corrupt frames
- Correct them using Error Correction Codes (ECC; also called Forward Error Correction FEC)
- Alternatively, detect errors and retransmit if necessary (ARQ)

Which should you use? What is the tradeoff?
- 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
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?