Network Delay

- Follow up on requirements from last week:
  "A new transatlantic cable (the first in 10 years) is going to be laid at the cost of $300M. The reason? To shave 6ms off the time to transmit packets from London to New York. The Hibernian Express will reduce the current transmission time — roughly 65 milliseconds — by less than ten percent. However, investors believe the financial community will be lining up to pay premium rates to use the new cable. The article suggests that a one-millisecond advantage could be worth $100M per year to a large hedge fund."

  — http://slashdot.org, September 13, 2011 @05:11AM

Link Layer: Implementation

- Implemented in “adapter”
  - E.g., PCMCIA card, Ethernet card
  - Typically includes: RAM, DSP chips, host bus interface, and link interface
Datalink Functions

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

Outline

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

How Encode?

- Seems obvious, why take time with this?

Why Encode?

- How many more ones?
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.

Non-Return to Zero (NRZ)

- 1 → high signal; 0 → low signal
- Used by Synchronous Optical Network (SONET)
- Long sequences of 1’s or 0’s can cause problems:
  - Sensitive to clock skew, i.e. hard to recover clock
  - DC bias hard to detect – low and high detected by difference from average voltage

Non-Return to Zero Inverted (NRZI)

- 1 → make transition; 0 → signal stays the same
- Solves the problem for long sequences of 1’s, but not for 0’s.

Clock Recovery

- When to sample voltage?
- Synchronized sender and receiver clocks
- Need easily detectible event at both ends
  - Signal transitions help resync sender and receiver
  - Need frequent transitions to prevent clock skew
  - SONET XOR’s bit sequence to ensure frequent transitions

http://yellowfourier.com/eyedia.html
Manchester Encoding
- Used by Ethernet
- 0=low to high transition, 1=high to low transition
- Transition for every bit simplifies clock recovery
- DC balance has good electrical properties
- Not very efficient
  - Doubles the number of transitions
  - Circuitry must run twice as fast

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

Other Encodings
- 8B/10B: Fiber Channel and Gigabit Ethernet
- 64B/66B: 10 Gbit Ethernet
- B8ZS: T1 signaling (bit stuffing)

Things to Remember
- Encoding necessary for clocking
- Lots of approaches
- Rule of thumb:
  - Little bandwidth → complex encoding
  - Lots of bandwidth → simple encoding
From Signals to Packets

- Analog Signal
- "Digital" Signal
- Bit Stream: \[0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1\]
- Packets: \[0100010101011100101010101011101110000001111010101110101010101101011010111001\]

Outline

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

Framing

- How do we differentiate the stream of bits into frames?
  \[0100010101011100101010111110000001111010101110101010101101011010111001\]
Out-of-band: E.g., 802.5

- 802.5/token ring uses 4b/5b
- Start delim & end delim are “illegal” codes

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 controls appear in data:
    - \*abc\*def \*abc\*def
    - Very common on serial lines, in editors, etc.
- Mark frames with special bit sequence
  - 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 \*111111101
    - must stuff a zero any time seven 1s appear:
    - 11111110 \*111111100
    - receiver unstuffs.

Ethernet Framing

- Preamble is 7 bytes of 10101010 (5 MHz square wave) followed by one byte of 10101011
  - With Manchester code, 10101 becomes 10 01 10 01 10, which looks like 1 00 11 00 11 0, which looks like 5 MHz square wave
- Allows receivers to recognize start of transmission after idle channel
Clock-Based Framing

- Used by SONET
- Fixed size frames (810 bytes)
- Look for start of frame marker that appears every 810 bytes
- Will eventually sync up

How avoid clock skew?

- Special bit sequences sent in first two chars of frame
- But no bit stuffing. Hmmm?
- Lots of transitions by xoring with special pattern (and hope for the best)

Outline

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

Error Coding

- Transmission process may introduce errors into a message.
  - 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

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.
- 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.
Example 1: Parity

- What is the minimum Hamming distance between two valid code words using a single parity bit?
- How many bit errors can always be detected with parity?
- If there are more bit errors, how often will they be detected?
- What is \( \lceil (D-1)/2 \rceil \) for parity?
- How many bit errors can be corrected with parity?

Examples

- A (4,3) parity code has \( D=2 \):
  - 0001 0010 0100 0111 1000 1011 1101 1110
  - (last bit is binary sum of previous 3, inverted - “odd parity”)
- A (7,4) code with \( D=3 \) (2ED, 1EC):
  - 0000000 0001101 0010111 0011010
  - 0100011 0101110 0110100 0111001
  - 1000110 1001011 1010001 1011100
  - 1100101 1101000 1110010 1111111
  - 1001111 corrects to 1001011
  - Note the inherent risk in correction; consider a 2-bit error resulting in 1001011 \( \rightarrow \) 1111011.
  - There are formulas to calculate the number of extra bits that are needed for a certain \( D \).

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:
  \[ 1 0 1 1 \]
  \[ X^4 + X^2 + X^1 + X^0 \]
- Sender and Receiver agree on a divisor polynomial of degree \( k \)
- Message of \( M \) bits \( \rightarrow \) 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 \( \geq k \) bits
Outline

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

Link Flow Control and Error Recovery

- Dealing with receiver overflow: flow control.
- Dealing with packet loss and corruption: error control.
- Meta-comment: these issues are relevant at many layers.
  - Link layer: sender and receiver attached to the same “wire”
  - End-to-end: transmission control protocol (TCP) - sender and receiver are the end points of a connection
- How can we implement flow control?
  - “You may send” (windows, stop-and-wait, etc.)
  - “Please shut up” (source quench, 802.3x pause frames, etc.)
  - Where are each of these appropriate?

A Naïve Protocol

- Sender simply sends to the receiver whenever it has packets.
- Potential problem: sender can outrun the receiver.
  - Receiver too slow, buffer overflow, ..
  - Not always a problem: receiver might be fast enough.

Adding Flow Control

- Stop and wait flow control: sender waits to send the next packet until the previous packet has been acknowledged by the receiver.
  - Receiver can pace the receiver
**Drawback: Performance**

Max Throughput = \( \frac{1 \text{ pkt}}{\text{Roundtrip Time}} \)

**Window Flow Control**

- Stop and wait flow control results in poor throughput for long-delay paths: packet size/roundtrip-time.
- Solution: receiver provides sender with a window that it can fill with packets.
  - The window is backed up by buffer space on receiver
  - Receiver acknowledges the a packet every time a packet is consumed and a buffer is freed

**Bandwidth-Delay Product**

Max Throughput = \( \frac{\text{Window Size}}{\text{Roundtrip Time}} \)

**Error Recovery**

- Two forms of error recovery
  - Error Correcting Codes (ECC)
  - Automatic Repeat Request (ARQ)
- ECC
  - Send extra redundant data to help repair losses
- ARQ
  - Receiver sends acknowledgement (ACK) when it receives packet
  - Sender uses ACKs to identify and resend data that was lost
- Which should we use? Why? When?
Error Recovery Example: Error Correcting Codes (ECC)

Two Dimensional Bit Parity:
Detect and correct single bit errors

<table>
<thead>
<tr>
<th></th>
<th>d_i,1</th>
<th>d_i,j</th>
<th>d_i,j+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>row parity &amp; column parity</td>
<td>d_1,1</td>
<td>d_1,j</td>
<td>d_1,j+1</td>
</tr>
<tr>
<td>d_2,1</td>
<td>d_2,j</td>
<td>d_2,j+1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>d_i,1</td>
<td>d_i,j</td>
<td>d_i,j+1</td>
<td></td>
</tr>
<tr>
<td>d_{i+1,1}</td>
<td>d_{i+1,j}</td>
<td>d_{i+1,j+1}</td>
<td></td>
</tr>
</tbody>
</table>

01101h parity error
11110h parity error
01110h parity error
01101h no errors
Parity error correctable single bit error

Stop and Wait
- Simplest ARQ protocol
- Send a packet, stop and wait until acknowledgement arrives
- Will examine ARQ issues later in semester

Recovering from Error

How to Recognize Retransmissions?
- Use sequence numbers
  - both packets and acks
- Sequence # in packet is finite → How big should it be?
  - For stop and wait?
- One bit – won’t send seq #1 until received ACK for seq #0
Issues with Window-based Protocol

- Receiver window size: # of out-of-sequence packets that the receiver can receive
- Sender window size: # of total outstanding packets that sender can send without acknowledged
- How to deal with sequence number wrap around?

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)

So far ...

Can connect two nodes

- ... But what if we want more nodes?

Wires for everybody!

Better Solutions: Datalink Architectures

- Point-Point with switches
- Media access control.
Outline

- Encoding
  - Digital signal to bits, e.g. NRZ, Manchester
  - Clock recovery/synchronization
- Framing
  - Bit stream to packets, e.g. character stuffing, bit stuffing, synchronous
  - Packet loss & corruption
    - Error detection, e.g. CRC, checksum, parity, Hamming Distance
    - Flow control, concept of windows
    - Loss recovery, e.g. error correction, ARQ

EXTRA SLIDES

Clock Based Framing: SONET

- SONET is the Synchronous Optical Network standard for data transport over optical fiber.
- One of the design goals was to be backwards compatible with many older telco standards.
- Beside minimal framing functionality, it provides many other functions:
  - operation, administration and maintenance (OAM) communications
  - synchronization
  - multiplexing of lower rate signals
  - multiplexing for higher rates
- In other words, really complicated!

Standardization History

- Process was started by divestiture in 1984.
  - Multiple telephone companies building their own infrastructure
  - SONET concepts originally developed by Bellcore.
- First standardized by ANSI T1X1 group for US.
- Later by CCITT and developed its own version.
A Word about Data Rates

- Bandwidth of telephone channel is under 4KHz, so when digitizing:
  \[ 8000 \text{ samples/sec} \times 8 \text{ bits} = 64 \text{Kbits/second} \]
- Common data rates supported by telcos in North America:
  - Modem: rate improved over the years
  - T1/DS1: 24 voice channels plus 1 bit per sample
    \[ (24 \times 8 + 1) \times 8000 = 1.544 \text{ Mbits/second} \]
  - T3/DS3: 28 T1 channels:
    \[ 7 \times 4 \times 1.544 = 44.736 \text{ Mbits/second} \]

Synchronous Data Transfer

- Sender and receiver are always synchronized.
- Frame boundaries are recognized based on the clock
- No need to continuously look for special bit sequences
- SONET frames contain room for control and data.
  - Data frame multiplexes bytes from many users
  - Control provides information on data, management, ...

How avoid clock skew?

- Special bit sequences sent in first two chars of frame
  - But no bit stuffing. Hmmm?
- Lots of transitions by xoring with special pattern (and hope for the best)

SONET Framing

- Base channel is STS-1 (Synchronous Transport System).
  - Takes 125 \( \mu \text{sec} \) and corresponds to 51.84 Mbps
  - 1 byte/frame corresponds to a 64 Kbs channel (voice)
  - Transmitted on an OC-1 optical carrier (fiber link)
- Standard ways of supporting slower and faster channels.
  - Support both old standards and future (higher) data rates
How Do We Support Lower Rates?

- 1 Byte in every consecutive frame corresponds to a 64 Kbit/second channel.
  - 1 voice call.
- Higher bandwidth channels hold more bytes per frame.
  - Multiples of 64 Kbit/second
- Channels have a “telecom” flavor.
  - Fixed bandwidth
  - Just data – no headers
  - SONET multiplexers remember how bytes on one link should be mapped to bytes on the next link
    - Byte 33 on incoming link 1 is byte 97 on outgoing link 7

125 µs

How Do We Support Higher Rates?

- Send multiple frames in a 125 µsec time slot.
- The properties of a channel using a single byte/ST-1 frame are maintained!
  - Constant 64 Kbit/second rate
  - Nice spacing of the byte samples
- Rates typically go up by a factor of 4.
- Two ways of doing interleaving.
  - Frame interleaving
  - Column interleaving
    - Concatenated version, i.e. OC-3c

125 µs

The SONET Signal Hierarchy

<table>
<thead>
<tr>
<th>Signal Type</th>
<th>line rate</th>
<th># of DS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS0 (POTS)</td>
<td>64 Kbs</td>
<td>1</td>
</tr>
<tr>
<td>DS1</td>
<td>1.544 Mbs</td>
<td>24</td>
</tr>
<tr>
<td>DS3</td>
<td>44.736 Mbs</td>
<td>672</td>
</tr>
<tr>
<td>OC-1</td>
<td>51.84 Mbs</td>
<td>672</td>
</tr>
<tr>
<td>OC-3</td>
<td>155 Mbs</td>
<td>2,016</td>
</tr>
<tr>
<td>OC-12</td>
<td>622 Mbs</td>
<td>8,064</td>
</tr>
<tr>
<td>STS-48</td>
<td>2.49 Gbs</td>
<td>32,256</td>
</tr>
<tr>
<td>STS-192</td>
<td>9.95 Gbs</td>
<td>129,024</td>
</tr>
<tr>
<td>STS-768</td>
<td>39.8 Gbs</td>
<td>516,096</td>
</tr>
</tbody>
</table>

STS-1 carries one DS-3 plus overhead

Using SONET in Networks

Add-drop capability allows soft configuration of networks, usually managed manually.
Self-Healing SONET Rings

SONET as Physical Layer

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 (ATM, HDCL)
CRC Example

Want:

\[ D \cdot 2^r \text{ XOR } R = nG \]

*equivalently:*

\[ D \cdot 2^r = \text{ nG XOR R} \]

*equivalently:*

if we divide \( D \cdot 2^r \) by \( G \),
want reminder \( R_b \)

\[ R = \text{ remainder} \left[ \frac{D \cdot 2^r}{G} \right] \]