Topics for this Week

- Data Link Layer
  - Data transmission over a single link
  - Framing and synchronization
  - Error control: detection/recovery

- Readings
  - Sections 3.1-3.4
Framing and Synchronization

- Problem: breaking bit stream into frames
  - Framing: recognizing bits on the wire as packets
- Must determine first and last bits of the frame
  - Framing and synchronization are closely related
- Typically implemented by a network adaptor
- Adaptor fetches/deposits frames out of/into host/switch memory
Framing Schemes

- Clock-based
  - Special bit pattern appears periodically

- Character/Byte-oriented
  - Character/Byte count
    - Problem: what if count field corrupted?
  - Starting and ending characters
    - STX (start of text) and ETX (end of text)
    - Problem: what if data contains these characters?
  - Character stuffing
    - Escape them by preceding them with DLE character
    - What if data contains DLE character?
Framing Schemes (cont’d)

- **Bit-oriented**
  - Each frame begins and ends with a special bit-sequence
    - Flag or preamble 01111110
  - **Bit-stuffing**
    - Sender: any time 5 consecutive 1’s in the body
      - Insert a 0
    - Receiver: should 5 consecutive 1’s arrive
      - If next bit is 0: remove
      - If next bits are 10: end-of-frame marker
      - If next bits are 11: error
Handling Errors

- Data can be corrupted during transmission
  - Bit values changed
- Frame includes additional information
  - Set by sender
  - Checked by receiver
- Error-detection vs error-correction
  - Both need redundant information
  - Detection: error exists or not.
  - Correction: repair if there was an error
- Statistical guarantee
Codewords and Hamming Distance

- **Codeword of** $n$-bit = $m$-bit data + $r$-bit check
  - Of $2^n$ possible codewords
    - Only $2^m$ are legal: used to represent data messages

- **Hamming distance**
  - Number of bit positions in which 2 codewords differ
  - If 2 codewords are at a Hamming distance of $d$
    - $d$ single-bit errors convert one to the other
  - **Hamming distance of complete code**
    - Distance between two closest legal codewords
Error Detecting and Correcting Codes

- Hamming distance and error detection/correction
  - To detect $d$ errors $\Rightarrow$ need $d+1$ distance code
  - To correct $d$ errors $\Rightarrow$ $2d+1$ distance code

- How many check/redundancy bits?
  - To detect single-bit error
    - 1 bit: even/odd parity
  - To correct a single-bit error in $m$-bit message
    - Need a minimum of $r$ bits such that
      - $(m + r + 1) \leq 2^r$
      - Example: 3-bit message needs 3-bit redundancy

- Correction or detection+retransmission?
An Error Correcting Code

<table>
<thead>
<tr>
<th>Char.</th>
<th>ASCII</th>
<th>Check bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1001000</td>
<td>00110010000</td>
</tr>
<tr>
<td>a</td>
<td>1100001</td>
<td>10111001001</td>
</tr>
<tr>
<td>R</td>
<td>1101101</td>
<td>11101010101</td>
</tr>
<tr>
<td>E</td>
<td>1101001</td>
<td>01101011001</td>
</tr>
<tr>
<td>E</td>
<td>1101110</td>
<td>01101010110</td>
</tr>
<tr>
<td>E</td>
<td>1100111</td>
<td>01111001111</td>
</tr>
<tr>
<td>e</td>
<td>0100000</td>
<td>10011000000</td>
</tr>
<tr>
<td>E</td>
<td>1100011</td>
<td>11111000111</td>
</tr>
<tr>
<td>E</td>
<td>1101111</td>
<td>10101011111</td>
</tr>
<tr>
<td>E</td>
<td>1100100</td>
<td>11111001100</td>
</tr>
<tr>
<td>e</td>
<td>1100101</td>
<td>00111000101</td>
</tr>
</tbody>
</table>

Order of bit transmission
Error Detection Techniques

- **Checksum**
  - Treat data as sequence of integers
  - Compute and send arithmetic sum
  - Handles multiple bit errors
  - Cannot handle all errors

- **Cyclic Redundancy Check**
  - Mathematical function of data
  - More complex to compute
  - Handles more errors
**Polynomial Error Detecting Code**

- Also known as *Cyclic Redundancy Code*
- Use modulo-2 (XOR) arithmetic with no carries
- Add \( r \) bits of redundancy to \( m \)-bit data
- Represent \( m \)-bit msg as \( m-1 \) degree polynomial
  - 10011010 corresponds to \( M(x) = x^7 + x^4 + x^3 + x^1 \)
- Choose divisor polynomial \( C(x) \) of degree \( r \)
  - E.g. \( C(x) = x^3 + x^2 + 1 \) (\( r = 3 \) and \( C(x) = 1101 \))
- Transmit polynomial \( T(x) \) that is divisible by \( C(x) \)
- Receiver gets polynomial \( T(x) + E(x) \)
  - \( E(x) \) is the polynomial corresponding to error bits
  - \( E(x) \) is 0 if no error
- How to compute \( T(x) \) from \( M(x) \) and \( C(x) \)?
Computation of CRC

Sender
- Multiply $M(x)$ by $x^r$
- Divide result $x^rM(x)$ by $C(x)$
- Subtract remainder from $x^rM(x)$
- The result is the polynomial $T(x)$

Receiver
- Receives $R(x) = T(x) + E(x)$
- Divides $R(x)$ by $C(x)$
- Remainder zero in two cases
  1. $E(x)$ is 0
  2. $E(x)$ is divisible by $C(x)$

Need to choose $C(x)$ such that 2$^{nd}$ case is rare
Frame: 1101011011
Generator: 10011
Message after appending 4 zero bits: 11010110110000

Transmitted frame: 11010110111110
Choice of $C(x)$

- Want to ensure $C(x)$ doesn’t divide $E(x)$
- We can detect
  - All single-bit errors if
    - $C(x)$ has at least 2 terms
  - All double-bit errors if
    - $C(x)$ can not be divided by $1+x^n$
  - Any odd number of errors if
    - $C(x)$ contains the factor $x+1$
  - Any “burst” error of length less than or equal to $r$ bits
  - Most burst errors of length greater than $r$ bits

- Ethernet and token ring networks use CRC-32
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$
Error Detection Summary

- To detect data corruption
  - Sender adds additional information to packet
  - Receiver checks
- Techniques
  - Parity bit
  - Checksum
  - Cyclic Redundancy Check
  - Provide statistical guarantees
Error Recovery

- Reliable delivery over unreliable channel
  - How to recover from corrupted/lost packets

- Error detection and retransmission
  - With acknowledgements and timeouts
  - Also called Automatic Repeat Request (ARQ)
  - Retransmission incurs round trip delay

- Error correcting codes
  - Also called Forward Error Correction (FEC)
  - No sender retransmission required
Stop-and-Wait ARQ

- Also called Alternating Bit Protocol
- Half-duplex-like communication
- Uses timers, seqno's (0/1), error detection bits
- Frames and ack's are numbered with
  - Alternating 0 and 1
- At each instant, the receiver
  - Expects a particular seqno
  - Discards frames with wrong/unexpected seqno
Stop-and-Wait ARQ: Algorithm

Sender: Proceeds in rounds
- Seqno alternating between 0 and 1, starting with 0
- At each round
  - Send a data frame with seqno of the round
  - Set retransmit timer
- Wait for ack with the seqno of the round
- If ack with correct seqno, flip it, go to next round
- If ack with incorrect seqno, ignore it
- If timer expires, resend the last frame

Receiver: Proceeds in rounds
- Ack seqno alternating between 0 and 1, starting with 0
- If expected seqno frame, send ack, flip seqno
- If unexpected seqno frame, discard it, resend last ack
Bandwidth-Delay Product and Throughput

Problem of Stop-and-Wait
- Can’t keep pipe full → low channel utilization

Issue: Large bandwidth-delay product
- Long propagation delay w.r.t. pkt transmission time

Example
- 1 Gbps, 1KB pkt → 8 μs to transmit a pkt
- Cross country propagation delay is 15 ms
- Utilization = fraction of time channel is busy
  - \( \frac{0.008}{2 \times 15 + 0.008} = 0.000266 \)
- Sender throughput is 266 Kbps with 1Gbps link
- Protocol (not channel capacity) constrains performance

Moral of the story
- Allow multiple unacknowledged pkts in-transit
Pipelined protocols

Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver

Two generic forms of pipelined protocols:

- Go-Back-N and Selective-Repeat
- Also known as sliding window protocols
Go-Back-N

Sender:
- k-bit seq # in pkt header
- “window” of up to N, consecutive unack’ed pkts allowed

- ACK(i): ACKs all pkts up to, including seq # i - “cumulative ACK”
  - may receive duplicate ACKs
- timer for each in-flight pkt
- timeout(i): retransmit pkt i and all higher seq # pkts in window
GBN in action

sender
send pkt0
send pkt1
send pkt2
send pkt3 (wait)
rcv ACK0
send pkt4
rcv ACK1
send pkt5
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5

receiver
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3
Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #’s
  - again limits seq #s of sent, unACKed pkts
Selective Repeat ARQ

- As in Go-Back-N
  - Packet sent when available up to window limit
  - Timer associated with each unACKed pkt

- Unlike Go-Back-N
  - Out-of-order (but otherwise correct) is ACKed
  - Receiver: buffers out-of-order packets
  - Sender: on timeout of packet k, retransmit just k

- Comments
  - More receiver buffering than Go-Back-N
  - More complicated buffer management by both sides
  - Saves bandwidth
    - No need to retransmit correctly received packets
Selective repeat in action

pkt0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 sent
0 1 2 3 4 5 6 7 8 9

pkt2 sent
0 1 2 3 4 5 6 7 8 9

pkt3 sent, window full
0 1 2 3 4 5 6 7 8 9

ACK0 rcvd, pkt4 sent
0 1 2 3 4 5 6 7 8 9

pkt2 timeout, pkt2 resent
0 1 2 3 4 5 6 7 8 9

ACK1 rcvd, pkt5 sent
0 1 2 3 4 5 6 7 8 9

pkt0 rcvd, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 rcvd, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9

pkt3 rcvd, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9

pkt4 rcvd, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9

pkt2 rcvd, deliver pkts 2, 3, 4
ACK2 sent
0 1 2 3 4 5 6 7 8 9

pkt5 rcvd, delivered, ACK5 sent
0 1 2 3 4 5 6 7 8 9
Segno Space and Window Size

- Suppose sequence numbers range from
  - 0 to MAXSEQNO
- How big the sliding window can be?
  - Under Go-Back-N?
    - MAXSEQNO+1 will not work, why?
  - What about Selective-Repeat?
Selective repeat: dilemma

Example:
- seq #’s: 0, 1, 2, 3
- window size=3

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?
Data Link Layer: Summary

- Framing and synchronization
- Error correction/detection codes
  - Parity/Checksum/CRC
- Error recovery protocols
  - Stop-and-Wait
  - Go-Back-N
  - Selective-Repeat
- Flow Control