Error Detection

Outline
- Parity
- Checksum
- CRC

2-Dimensional Parity

<table>
<thead>
<tr>
<th>Data</th>
<th>Parity byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101001</td>
<td>1</td>
</tr>
<tr>
<td>1101001</td>
<td>0</td>
</tr>
<tr>
<td>1011110</td>
<td>1</td>
</tr>
<tr>
<td>0001110</td>
<td>1</td>
</tr>
<tr>
<td>0110100</td>
<td>1</td>
</tr>
<tr>
<td>1011111</td>
<td>0</td>
</tr>
<tr>
<td>1111011</td>
<td>0</td>
</tr>
</tbody>
</table>
Internet Checksum Algorithm

- View message as a sequence of 16-bit integers; sum using 16-bit ones-complement arithmetic; take ones-complement of the result.

```c
u_short cKsum(u_short *buf, int count) {
    register u_long sum = 0;
    while (count--) {
        sum += *buf++;
        if (sum & 0xFFFF0000) {
            /* carry occurred, so wrap around */
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

Cyclic Redundancy Check

- Add $k$ bits of redundant data to an $n$-bit message
  - want $k << n$
  - e.g., $k = 32$ and $n = 12,000$ (1500 bytes)
- Represent $n$-bit message as $n$-1 degree polynomial
  - e.g., MSG=10011010 as $M(x) = x^7 + x^4 + x^3 + x^1$
- Let $k$ be the degree of some divisor polynomial
  - e.g., $C(x) = x^3 + x^2 + 1$
CRC (cont)

- Transmit polynomial $P(x)$ that is evenly divisible by $C(x)$
  - shift left $k$ bits, i.e., $M(x)x^k$
  - subtract remainder of $M(x)x^k / C(x)$ from $M(x)x^k$
- Receiver polynomial $P(x) + E(x)$
  - $E(x) = 0$ implies no errors
- Divide $(P(x) + E(x))$ by $C(x)$; remainder zero if:
  - $E(x)$ was zero (no error), or
  - $E(x)$ is exactly divisible by $C(x)$

Selecting $C(x)$

- 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 $C(x)$ contains a factor with at least three terms
- Any odd number of errors, as long as $C(x)$ contains the factor $(x + 1)$
- Any ‘burst’ error (i.e., sequence of consecutive error bits) for which the length of the burst is less than $k$ bits.
- Most burst errors of larger than $k$ bits can also be detected
- See Table 2.5 on page 96 for common $C(x)$
Hardware Implementation

Acknowledgements & Timeouts

(a) (c)
(b) (d)
Stop-and-Wait

• Problem: keeping the pipe full
• Example
  – 1.5Mbps link × 45ms RTT = 67.5Kb (8KB)
  – 1KB frames implies 1/8th link utilization
Sliding Window

- Allow multiple outstanding (un-ACKed) frames
- Upper bound on un-ACKed frames, called \textit{window}

\begin{itemize}
  \item Assign sequence number to each frame (\textit{SeqNum})
  \item Maintain three state variables:
    \begin{itemize}
      \item send window size (\textit{SWS})
      \item last acknowledgment received (\textit{LAR})
      \item last frame sent (\textit{LFS})
    \end{itemize}
  \item Maintain invariant: \textit{LFS} - \textit{LAR} <= \textit{SWS}
  \item Advance \textit{LAR} when ACK arrives
  \item Buffer up to \textit{SWS} frames
\end{itemize}
**SW: Receiver**

- Maintain three state variables
  - receive window size (RWS)
  - largest frame acceptable (LFA)
  - last frame received (LFR)
- Maintain invariant: \( \text{LFA} - \text{LFR} \leq \text{RWS} \)

  ![Frame Sequence Diagram]

- Frame \( \text{SeqNum} \) arrives:
  - if \( \text{LFR} < \text{SeqNum} \leq \text{LFA} \) accept
  - if \( \text{SeqNum} < \text{LFR} \) or \( \text{SeqNum} > \text{LFA} \) discarded
- Send cumulative ACKs

---

**Sequence Number Space**

- \( \text{SeqNum} \) field is finite; sequence numbers wrap around
- Sequence number space must be larger than number of outstanding frames
- \( \text{SWS} \leq \text{MaxSeqNum} - 1 \) is not sufficient
  - suppose 3-bit \( \text{SeqNum} \) field (0..7)
  - \( \text{SWS}=\text{RWS}=7 \)
  - sender transmit frames 0..6
  - arrive successfully, but ACKs lost
  - sender retransmits 0..6
  - receiver expecting 7, 0..5, but receives second incarnation of 0..5
- \( \text{SWS} < \frac{(\text{MaxSeqNum}+1)}{2} \) is correct rule
- Intuitively, \( \text{SeqNum} \) “slides” between two halves of sequence number space
Concurrent Logical Channels

• Multiplex 8 logical channels over a single link
• Run stop-and-wait on each logical channel
• Maintain three state bits per channel
  – channel busy
  – current sequence number out
  – next sequence number in
• Header: 3-bit channel num, 1-bit sequence num
  – 4-bits total
  – same as sliding window protocol
• Separates reliability from order