Error Detection and Reliable Transmission
EECS 122: Lecture 24

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley
Today

- Focus on a Link Layer function
  - Bits need to be sent over a link in frames
  - Bits may be corrupted in transit
    - E.g. send 111 but 110 is received

- Issues:
  - How can the destination tell that bits have been corrupted?
    - Error Detection
  - What schemes will ensure that the correct bits are received?
    - Retransmission Schemes (already covered this)
      - Stop and Wait
      - Go back n
    - Error Correcting Codes
Two hosts, want to communicate over a bidirectional link

But...
• Communication must be reliable
• Bits generated asynchronously

Example: Manchester Encoding

Physical Layer

RS232-C Interface Modem Modem RS232-C Interface
Interface wires Physical Medium Interface wires

Synchronous unreliable bit pipe

Physical Medium ? Virtual Bit Pipe

1 0 X 1 1 X

Synchronous
Data Link Control Layer

Two hosts, want to communicate asynchronously and reliably over a bidirectional link

1. Frame the data
2. Number frames
3. Send CRC
4. Retransmit if required

Example: SONET
Error Detection at higher layers

- Overlays: The “link” layer can actually involve many physical links
- When a network is considered to be reliable, it may make sense to do error detection end-to-end rather than on every link
- When a link is unreliable it makes sense to do error detection at layer 3 so that routers can check for bad headers without having to worry about the payload
Error detection

- The frame consists of a header and payload

<table>
<thead>
<tr>
<th>Header</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>n bits</td>
<td></td>
</tr>
</tbody>
</table>

- If all n-bit strings are valid payload strings, the error detection must occur in the header bits

- Break the header into two parts

<table>
<thead>
<tr>
<th>FH</th>
<th>Payload</th>
<th>EDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>n bits</td>
<td>k bits</td>
<td></td>
</tr>
</tbody>
</table>

- Error detecting code part contains bits that add redundancy
Error Detecting Codes

- **Goals:**
  - Reduce overhead, i.e., reduce the number of redundancy bits
  - Increase the number and the type of bit error patterns that can be detected

- **Examples:**
  - Even Parity
  - Rectangular Codes
  - Cyclic Redundancy Check (CRC)
Even Parity Check

- EDC field has 1 bit
- Sender: If number of 1’s in payload is odd, the check bit is 1, else it is 0
- Receiver: Accept the packet if payload number of 1’s match the value of the check bit.
- Can detect an odd number of bit errors
Two-dimensional Parity

- Add one extra bit to a 7-bit code such that the number of 1’s in the resulting 8 bits is even (for even parity, and odd for odd parity)
- Add a parity byte for the packet
- Example: five 7-bit character packet, even parity

<table>
<thead>
<tr>
<th>Packet</th>
<th>Parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110100</td>
<td>1</td>
</tr>
<tr>
<td>1011010</td>
<td>0</td>
</tr>
<tr>
<td>0010110</td>
<td>1</td>
</tr>
<tr>
<td>1110101</td>
<td>1</td>
</tr>
<tr>
<td>1001011</td>
<td>0</td>
</tr>
<tr>
<td>1000110</td>
<td>1</td>
</tr>
</tbody>
</table>
How Many Errors Can you Detect?

- All 1-bit errors
- Example:

```
0110100
1011010
0000110
1110101
1001011
```

Can detect AND correct 1-bit errors
How Many Errors Can you Detect?

- All 2-bit errors
- Example:

```
0110100  1
1011010  0
0000111  1
1110101  1
1001011  0
1000110  1
```

error bits
odd number of 1's on column
How Many Errors Can you Detect?

- All 3-bit errors
- Example:

```
0110100
1011010
0000111
1100101
1001011
1000110
```

1 0 1 1 1
0 0 0 1 1
1 1 0 0 1
0 1 0 1 1
1 0 0 1 1
0 1

error bits
odd number of 1’s on column
How Many Errors Can you Detect?

- Most 4-bit errors
- Example of 4-bit error that is not detected:

```
  0110100   1
  1011010   0
  0000111   1
  1100100   1
  1001011   0
  1000110   1
```

error bits
Burst Errors

- Often errors occur in bursts, i.e. successive bits are corrupted.
  - Many detecting/correcting schemes fail
- Use an interleaving trick…
- Suppose we 4 code words to send
  - 0101001,0000000,1111111,1010101
- Send position 1 of each codeword, then position 2 of each codeword etc.
  - 0011 1010 0011 1010 0010 0010 1011
- A burst of 4 errors looks like 4 single bit errors to each of the code words!
**Cyclic Redundancy Check Codes (CRC)**

- Very popular error detection code
- Efficient Implementations in HW and SW
- Represent the payload as a polynomial
- The EDC bits are generated by dividing the payload polynomial by another special polynomial called the generator.
- The generator can be picked to ensure that a variety of different errors are detected.
Cyclic Redundancy Check (CRC)

- Represent a n-bit payload by an polynomial $P(x)$
  - E.g., $10101101 \rightarrow P(x) = x^7 + x^5 + x^3 + x^2 + x^0$

- Choose a generator k-degree polynomial $G(x)$

- Compute remainder $R(x)$ of $P(x)*x^k / G(x)$
  - Note that $P(x)*x^k$ represents the the payload shifted to the left by k places
  - E.g. If $P(x)=x^7 + x^5 + x^3 + x^2 + x^0$ and $k=2$ then
    - $P(x)*x^k = x^9 + x^7 + x^5 + x^4 + x^2 \rightarrow 1010110100$

- The EDC bits are derived from $-R(x)$
  - A-B in binary is a bitwise XOR
  - E.g. If $k=2$ and $R(x)=1$ then EDC bits are $00 - 01 = 01$
CRC

- \( T(x) = M(x)^k x^k - R(x) \)
  - \( T(x) \) is divisible by \( C(x) \)
  - First \( n \) coefficients of \( T(x) \) represent \( M(x) \)
  - E.g. If \( P(x) = x^7 + x^5 + x^3 + x^2 + x^0 \), \( R(x) = x + 1 \), \( k = 2 \)
    then \( T(x) = x^9 + x^7 + x^5 + x^4 + x^2 + 1 \)

- **Sender:**
  - Compute and send \( T(x) \), i.e., the coefficients of \( T(x) \)

- **Receiver:**
  - Let \( T'(x) \) is what he gets
  - If \( C(x) \) divides \( T'(x) \) \( \Rightarrow \) no errors; otherwise errors
Computing $R(x)$

- Compute the remainder $R(x)$ of $P(x)\times x^k / G(x)$
- $T(x) = P(x)\times x^k - R(x)$

Example: send packet 110111, assume $G(x) = 101$
- $k = 2$, $P(x)\times x^k \Rightarrow 11011100$
- Compute $R(x)$

\[
\begin{array}{c|cccc}
 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
\hline
1 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
\hline
& 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
\hline
\end{array}
\]

- $T(x) = P(x)\times x^k - R(x) \Rightarrow 11011100 \text{xor} 1 = 11011101$
Check at the receiver

$101 \) 11011101$

$101$

$111$

$101$

$101$

$101$

$101$

$101$

$0 \quad \text{No errors}$
Properties of CRCs

- Detect all single-bit errors if coefficients of $x^k$ and $x^0$ of $G(x)$ are one
- Detect all double-bit errors, if $G(x)$ has a factor with at least three terms
- Detect all number of odd errors, if $G(x)$ contains factor $(x+1)$
- Detect all burst of errors smaller than $k$ bits
What’s going on?

- Think of the combination of the \( n \) payload bits and the \( k \) EDC bits as being a \( n+k \) bit codeword.
- For any error correcting scheme, not all \( n+k \) bit strings will be valid code words.
- Errors can be detected if and only if the received string is not a valid codeword.
  - Example: Even parity check only detects an odd number of bit errors.
Hamming Distance

- Given codewords A and B, the Hamming distance between them is the number of bits in A that need to be flipped to turn it into B
  - Example $H(011101,000000)=4$
- If all codewords are at least $d$ Hamming distance apart, then up to $d-1$ bit errors can be detected
Error Correction

- If all the codewords are at least a hamming distance of $2E+1$ apart then up to $E$ bit errors can be detected
  - Just pick the codeword closest to the one received!
- How many bits of EDC are required to correct $d$ errors when there are $n$ bits in the payload?
- Example: $d=1$: Suppose $n=3$. Then any payload can be transformed into 3 other payload strings (000 into 001, 010 or 100).
  - Need at least two bits in the ECD to differentiate between 4 possibilities
  - In general need at least $k = \log_2(n+1)$ bits
  - A scheme that is optimal is called a perfect parity code
Perfect Parity Codes

- Consider a code word of n+k bits
  - $b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} \ldots$
- Parity bits are in positions $2^0, 2^1, 2^2, 2^3, 2^4 \ldots$
  - $b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} \ldots$
- A parity bit in position $2^h$, checks all data bits $b_p$ such that if you write out $p$ in binary, the $h^{th}$ place is a one.
- Example: $b_1$ is a parity bit. It is applied over all data bits $b_p$ such that the zero$^{th}$ position of $p$ is a one, i.e. $p$ is odd
A (7,4)-Perfect Parity Code

- n=4, k=3
  - Since \( \log(n+1) = 2.322 \), a code with \( k=3 \) that corrects one error is "perfect"
- Supposing the 6\(^{th}\) bit is corrupted in a codeword. The binary rep of 6 is 110, so the parity bits at positions 2 and 4 would fail. There is no other case when these two bits would fail together
  - Note 2+4 is 6
- Suppose the 5\(^{th}\) bit fails. 5\( \rightarrow \)101 so the parity bits at positions 1 and 4 would fail
  - Note 1+4 is 5
- Suppose the 4\(^{th}\) bit is corrupted. 4\( \rightarrow \) 100 so the parity bit at position 4 would fail
  - Corrupted Parity bits can be identified as well
Reliable Transmission

- Problem: obtain correct information once errors are detected
- Solutions:
  - Use error correction codes (e.g. perfect parity codes)
  - Use retransmission (we have studied this already)
- Algorithmic challenges:
  - Achieve high link utilization, and low overhead
Error correction or Retransmission?

- Error Correction requires a lot of redundancy
  - Wasteful if errors are unlikely
- Retransmission strategies are more popular
  - As links get reliable this is just done at the transport layer
- Error correction is useful when retransmission is costly (satellite links, multicast)