Direct Link Networks – Error Detection and Correction

Reading: Peterson and Davie, Chapter 2
Error Detection

- Encoding translates symbols to signals
- Framing demarcates units of transfer
- Error detection validates correctness of each frame
Error Detection

- Adds redundant information that checks for errors
  - And potentially fix them
  - If not, discard packet and resend
- Occurs at many levels
  - Demodulation of signals into symbols (analog)
  - Bit error detection/correction (digital)—our main focus
    - Within network adapter (CRC check)
    - Within IP layer (IP checksum)
    - Within some applications
Error Detection

- Analog Errors
  - Example of signal distortion

- Hamming distance
  - Parity and voting
  - Hamming codes

- Error bits or error bursts?

- Digital error detection
  - Two-dimensional parity
  - Checksums
  - Cyclic Redundancy Check (CRC)
Analog Errors

- Consider RS-232 encoding of character ‘Q’
- Assume idle wire (-15V) before and after signal
RS-232 Encoding of 'Q'

start 1 1 0 0 0 0 0 1 stop
Encoding isn’t perfect

Example with bandwidth = baud rate

![Graph showing voltage levels with 1s and 0s at different points, illustrating the concept of encoding in communication systems.]
Encoding isn’t perfect

Example with bandwidth = baud rate/2
Symbols

possible binary voltage encoding symbol neighborhoods and erasure region

possible QAM symbol neighborhoods in green; all other space results in erasure
Digital error detection and correction

- **Input**: decoded symbols
  - Some correct
  - Some incorrect
  - Some erased

- **Output**:
  - Correct blocks (or codewords, or frames, or packets)
  - Erased blocks
Error Detection Probabilities

Definitions

- \( P_b \) : Probability of single bit error (BER)
- \( P_1 \) : Probability that a frame arrives with no bit errors
- \( P_2 \) : While using error detection, the probability that a frame arrives with one or more undetected errors
- \( P_3 \) : While using error detection, the probability that a frame arrives with one or more detected bit errors but no undetected bit errors
Error Detection Probabilities

- With no error detection

\[ P_1 = (1 - P_b)^F \]

\[ P_2 = 1 - P_1 \]

\[ P_3 = 0 \]

- No bit errors
- Undetected errors
- Detected errors

- \( F = \) Number of bits per frame

Single bit error
Error Detection Process

- **Transmitter**
  - For a given frame, an error-detecting code (check bits) is calculated from data bits
  - Check bits are appended to data bits

- **Receiver**
  - Separates incoming frame into data bits and check bits
  - Calculates check bits from received data bits
  - Compares calculated check bits against received check bits
  - Detected error occurs if mismatch
Parity

- Parity bit appended to a block of data
- Even parity
  - Added bit ensures an even number of 1s
- Odd parity
  - Added bit ensures an odd number of 1s

Example
- 7-bit character 1110001
- Even parity 1110001 0
- Odd parity 1110001 1
Parity: Detecting Bit Flips

1-bit error detection with parity
- Add an extra bit to a code to ensure an even (odd) number of 1s
- Every code word has an even (odd) number of 1s

<table>
<thead>
<tr>
<th>Valid code words</th>
<th>Parity Encoding: White – invalid (error)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 11</td>
<td>011 111</td>
</tr>
<tr>
<td>00 10</td>
<td>010 110</td>
</tr>
<tr>
<td></td>
<td>000 100</td>
</tr>
<tr>
<td></td>
<td>001 101</td>
</tr>
</tbody>
</table>
Voting: Correcting Bit Flips

- 1-bit error correction with voting
  - Every codeword is transmitted n times
  - Codeword is 3 bits long

Valid code words:
0 \rightarrow 1

Voting:
- White – correct to 1
- Blue - correct to 0
Voting: 2-bit Erasure Correction

- Every code word is copied 3 times

2-erasure planes in green remaining bit not ambiguous

cannot correct 1-error and 1-erasure
Hamming Distance

- The Hamming distance between two code words is the minimum number of bit flips to move from one to the other.
  - Example:
  - 00101 and 00010
  - Hamming distance of 3
Minimum Hamming Distance

- The minimum Hamming distance of a code is the minimum distance over all pairs of codewords
  - Minimum Hamming Distance for parity
    - 2
  - Minimum Hamming Distance for voting
    - 3
Coverage

- **N-bit error detection**
  - No code word changed into another code word
  - Requires Hamming distance of $N+1$

- **N-bit error correction**
  - N-bit neighborhood: all codewords within $N$ bit flips
  - No overlap between N-bit neighborhoods
  - Requires hamming distance of $2N+1$
Hamming Codes

- Linear error-correcting code
- Named after Richard Hamming
- Simple, commonly used in RAM (e.g., ECC-RAM)
- Can detect up to 2-bit errors
- Can correct up to 1-bit errors
Hamming Codes

- **Construction**
  - number bits from 1 upward
  - powers of 2 are check bits
  - all others are data bits
  - Check bit \( j \): XOR of all \( k \) for which \( (j \text{ AND } k) = j \)

- **Example:**
  - 4 bits of data, 3 check bits

```
  1  2  3  4  5  6  7
  C_1 C_2 D_3 C_4 D_5 D_6 D_7
```

\[
C_1 \oplus C_2 \oplus D_3 \oplus C_4 \oplus D_5 \oplus D_6 \oplus D_7 = \begin{cases} 
0 & \text{if correct} \\
1 & \text{if incorrect}
\end{cases}
\]
Hamming Codes

- **Construction**
  - number bits from 1 upward
  - powers of 2 are check bits
  - all others are data bits
  - Check bit \( j \): XOR of all \( k \) for which \((j \ AND \ k) = j\)

- **Example:**
  - 4 bits of data, 3 check bits

```
 1  2  3  4  5  6  7
C_1 C_2 D_3 C_4 D_5 D_6 D_7
```
Hamming Codes

Construction
- number bits from 1 upward
- powers of 2 are check bits
- all others are data bits
- Check bit $j$: XOR of all $k$ for which $(j \text{ AND } k) = j$

Example:
- 4 bits of data, 3 check bits

```
C_1 C_2 D_3 C_4 D_5 D_6 D_7
1 2 3 4 5 6 7
```
Hamming Codes
Cost of a Hamming code

- $n$ bit codewords
  - Assume $n = 2^m - 1$ for some $m$
- $k$ bits of message
  - $k = n - \log_2(n+1)$
- Can we do any better for correcting 1-bit errors?
Minimum bits to correct 1 error

- $n$ bit codewords, $k$ bits of message
- If we can correct 1 bit error, what do we know?
  - The $k$ message bits
  - Which bit was erroneous
- How many **bits of information** do we know?
  - $k$ for the message, obviously
  - $\log_2(n)$ bits to identify which of $n$ bits was erroneous
- #bits we know $\leq$ # bits we get, so
  - $k + \log_2(n) \leq n$
  - $k \leq n - \log_2(n)$
Minimum bits to correct 1 error: improved bound

- $n$ bit codewords; $k$ bits of message
- If we can correct 1 bit error, what do we know?
  - The $k$ message bits
  - Which of $n+1$ outcomes: all correct, or one of $n$ bits wrong
- How many **bits of information** do we know?
  - $k$ for the message, obviously
  - $\log_2(n+1)$ bits to identify which of $n+1$ outcomes we’re in
- #bits we know $\leq$ # bits we get, so
  - $k + \log_2(n+1) \leq n$
  - $k \leq n - \log_2(n+1)$
- Hamming codes are **perfect codes**!
Minimum bits to correct 1 error: alternate bound

- $n$ bit codewords; $k$ bits of message

$n$ codewords within Hamming distance 1
Minimum bits to correct 1 error: alternate bound

- $n$ bit codewords; $k$ bits of message
- $2^n$ total codewords
- Each valid codeword “uses up” $1 + n$ of them
- So, at most $2^n / (n+1)$ valid codewords
- $2^k \leq 2^n / (n+1)$
- So, $k \leq n - \log_2(n+1)$.
What are we trying to handle?

- Worst case errors
  - We solved this for 1 bit error
  - Can generalize, but will get expensive for more bit errors
- Probability of error per bit
  - Flip each bit with some probability, independently of others
- Burst model
  - Probability of back-to-back bit errors
  - Error probability dependent on adjacent bits
  - Value of errors may have structure
- Why assume bursts?
  - Appropriate for some media (e.g., radio)
  - Faster signaling rate enhances such phenomena
Digital Error Detection Techniques

- Two-dimensional parity
  - Detects up to 3-bit errors
  - Good for burst errors

- IP checksum
  - Simple addition
  - Simple in software
  - Used as backup to CRC

- Cyclic Redundancy Check (CRC)
  - Powerful mathematics
  - Tricky in software, simple in hardware
  - Used in network adapter
Two-Dimensional Parity

- Use 1-dimensional parity
  - Add one bit to a 7-bit code to ensure an even/odd number of 1s

- Add 2nd dimension
  - Add an extra byte to frame
    - Bits are set to ensure even/odd number of 1s in that position across all bytes in frame

- Comments
  - Catches all 1-, 2- and 3-bit and most 4-bit errors
### Two-Dimensional Parity

A two-dimensional parity is a method of error detection and correction used in digital logic and computer science. It involves a set of binary numbers arranged in a grid, where the parity of each row and column is calculated. The parity is typically checked by adding all the bits in a row or column and ensuring the sum is even (even parity) or odd (odd parity). If the sum is not as expected, an error is detected.

Here is an example of a 2x2 two-dimensional parity:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The parity check is done by counting the number of 1s in each row and column. For example, the first row has a single 1, so the parity is 0 (odd parity). The first column has a single 1, so the parity is 0 (even parity).

In this example, there is an additional bit for the parity of the entire grid, which is calculated by adding all the bits and checking if the sum is even or odd.
Internet Checksum

- Idea
  - Add up all the words
  - Transmit the sum
  - Use 1’s complement addition on 16bit codewords
- Example
  - Codewords: -5, -3
  - 1’s complement binary: 1010, 1100
  - 1’s complement sum 1000

- Comments
  - Small number of redundant bits
  - Easy to implement
  - Not very robust
  - Eliminated in IPv6
IP Checksum

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);
}
Main Goal: Check the Data!

- $n$ data bits

- Hash function

- $k$ pseudorandom check bits
Main Goal: Check the Data!

- In any code, what fraction of codewords are valid?
  - \(1/2^k\)
- Ideal (random) hash function:
  - Any change in input produces an output that’s essentially random
  - So any error would be detected with probability \(1 – 2^{-k}\)
- Checksum: not close to ideal
- CRC: better
Simplified CRC-like protocol using regular integers

- **Basic idea**
  - Both endpoints agree in advance on divisor value $C = 3$
  - Sender wants to send message $M = 10$
  - Sender computes $X$ such that $C$ divides $10M + X$
  - Sender sends codeword $W = 10M + X$
  - Receiver receives $W'$ and checks whether $C$ divides $W'$
    - If so, then probably no error
    - If not, then error
Simplified CRC-like protocol using regular integers

Intuition

- If $C$ is large, it’s unlikely that bits are flipped exactly to land on another multiple of $C$.
- CRC is vaguely like this, but uses polynomials instead of numbers.
Cyclic Redundancy Check (CRC)

- **Given**
  - Message $M = 10011010$
  - Represented as Polynomial $M(x)$
    $$M(x) = 1 \times x^7 + 0 \times x^6 + 0 \times x^5 + 1 \times x^4 + 1 \times x^3 + 0 \times x^2 + 1 \times x + 0$$
    $$= x^7 + x^4 + x^3 + x$$

- **Select a divisor polynomial** $C(x)$ **with degree** $k$
  - Example with $k = 3$:
    - $C(x) = x^3 + x^2 + 1$
    - Represented as 1101

- **Transmit a polynomial** $P(x)$ **that is evenly divisible** by $C(x)$
  - $P(x) = M(x) \times x^k + k$ check bits

**How can we determine these** $k$ **bits?**
Properties of Polynomial Arithmetic

- Coefficients are modulo 2
  \[(x^3 + x) + (x^2 + x + 1) = \ldots\]
  \[\ldots x^3 + x^2 + 1\]
  \[(x^3 + x) - (x^2 + x + 1) = \ldots\]
  \[\ldots x^3 + x^2 + 1\text{ also!}\]

- Addition and subtraction are both xor!

- Need to compute \( R \) such that \( C(x) \) divides \( P(x) = M(x) \cdot x^k + R(x) \)

- So \( R(x) = \) remainder of \( M(x) \cdot x^k / C(x) \)
  - Will find this with polynomial long division
Polynomial arithmetic

- **Divisor**
  - Any polynomial $B(x)$ can be divided by a polynomial $C(x)$ if $B(x)$ is of the same or higher degree than $C(x)$

- **Remainder**
  - The remainder obtained when $B(x)$ is divided by $C(x)$ is obtained by subtracting $C(x)$ from $B(x)$

- **Subtraction**
  - To subtract $C(x)$ from $B(x)$, simply perform an XOR on each pair of matching coefficients

- For example: $(x^3+1)/(x^3+x^2+1) = \square$
CRC - Sender

- **Given**
  - \( M(x) = 10011010 \) = \( x^7 + x^4 + x^3 + x \)
  - \( C(x) = 1101 \) = \( x^3 + x^2 + 1 \)

- **Steps**
  - \( T(x) = M(x) \times x^k \) (add zeros to increase deg. of \( M(x) \) by \( k \))
  - Find remainder, \( R(x) \), from \( T(x)/C(x) \)
  - \( P(x) = T(x) - R(x) \Rightarrow M(x) \) followed by \( R(x) \)

- **Example**
  - \( T(x) = 10011010000 \)
  - \( R(x) = 101 \)
  - \( P(x) = 10011010101 \)
CRC - Receiver

- Receive Polynomial $P(x) + E(x)$
  - $E(x)$ represents errors
  - $E(x) = 0$, implies no errors
- Divide $(P(x) + E(x))$ by $C(x)$
  - If result = 0, either
    - No errors ($E(x) = 0$, and $P(x)$ is evenly divisible by $C(x)$)
    - $(P(x) + E(x))$ is exactly divisible by $C(x)$, error will not be detected
  - If result = 1, errors.
CRC – Example Encoding

\[ C(x) = x^3 + x^2 + 1 = 1101 \quad \text{Generator} \]
\[ M(x) = x^7 + x^4 + x^3 + x = 10011010 \quad \text{Message} \]

1101 \quad 10011010000 \quad \text{Message plus k zeros}

\[ \text{Remainder} \quad m \mod c \]

Result:
Transmit message followed by remainder:
\[ 10011010101 \]
CRC – Example Decoding – No Errors

C(x) = x^3 + x^2 + 1 = 1101 Generator
P(x) = x^{10} + x^7 + x^6 + x^4 + x^2 + 1 = 10011010101 Received Message

\[
\begin{array}{c}
\text{k + 1 bit check sequence c, equivalent to a degree-k polynomial}\\
1101
\end{array}
\]

\[
\begin{array}{c}
\text{Received message, no errors}\\
100110101011101
\end{array}
\]

Result:
CRC test is passed

\[
\begin{array}{c}
\text{Remainder}\\
m \mod c
\end{array}
\]

\[
\begin{array}{c}
\text{0}
\end{array}
\]
CRC – Example Decoding – with Errors

C(x) = \( x^3 + x^2 + 1 \) = 1101
P(x) = \( x^{10} + x^7 + x^5 + x^4 + x^2 + 1 \) = 10010110101

Result:
CRC test failed
CRC Error Detection

Properties
- Characterize error as $E(x)$
- Error detected unless $C(x)$ divides $E(x)$
  - (i.e., $E(x)$ is a multiple of $C(x)$)
Example of Polynomial Multiplication

- Multiply
  - 1101 by 10110
  - \( x^3 + x^2 + 1 \) by \( x^4 + x^2 + x \)

This is a multiple of \( c \), so that if errors occur according to this sequence, the CRC test would be passed.
On Polynomial Arithmetic

- The use of polynomial arithmetic is a fancy way to think about addition with no carries. It also helps in the determination of a good choice of $C(x)$
  - A non-zero vector is not detected if and only if the error polynomial $E(x)$ is a multiple of $C(x)$

**Implication**

- Suppose $C(x)$ has the property that $C(1) = 0$ (i.e. $(x + 1)$ is a factor of $C(x)$)
- If $E(x)$ corresponds to an undetected error pattern, then it must be that $E(1) = 0$
- Therefore, any error pattern with an odd number of error bits is detected
CRC Error Detection

- **What errors can we detect?**
  - All single-bit errors, if $x^k$ and $x^0$ have non-zero coefficients
  - All double-bit errors, if $C(x)$ has at least three terms
  - All odd bit errors, if $C(x)$ contains the factor $(x + 1)$
  - Any bursts of length $< k$, if $C(x)$ includes a constant term
  - Most bursts of length $\geq k$
# Common Polynomials for C(x)

<table>
<thead>
<tr>
<th>CRC</th>
<th>C(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC-8</td>
<td>$x^8 + x^2 + x^1 + 1$</td>
</tr>
<tr>
<td>CRC-10</td>
<td>$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$</td>
</tr>
<tr>
<td>CRC-12</td>
<td>$x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$</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>$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$</td>
</tr>
</tbody>
</table>
Odd-bit detection

- $E(x)$ must contain an odd number of terms
- However, no polynomial with an odd number of terms is divisible by $(x + 1)$
- Make $(x + 1)$ a factor of $C(x)$!
Another way of thinking about it

- Assume $C(x) = C'(x)(x + 1)$
- Implies $P(x) = C(x)f(x) = (x + 1)C'(x)f(x)$
- $P(1) = (0)C'(1)f(1) = 0$
- What if $P(x) + E(x)$ received?
  - Detected if $P(1) + E(1) = E(1)$ is not 0
  - $E(1)$ is 0 iff $E(x)$ has an even number of terms
    - Can’t detect even # of bit error
    - Can detect odd number of bit errors
CRC: Detecting Burst Errors of Less than k Bits

- Assume that \( C(x) = x^k + ... + 1 \)
- Write \( E(x) = x^i (x^{k-1} + ... + 1) \)
  - All factors are present
  - \( i \) represents the location of the first bit error from the right side of the received frame
- If \( C(x) \) contains \( x^0 \), it cannot have \( x^i \) as a factor
  - The degree of the rest of \( E(x) \) is less than the degree of \( C(x) \) and so the remainder must be non-zero
Error Detection vs. Error Correction

- Detection
  - Pro: Overhead only on messages with errors
  - Con: Cost in bandwidth and latency for retransmissions

- Correction
  - Pro: Quick recovery
  - Con: Overhead on all messages

What should we use?
- Correction if retransmission is too expensive
- Correction if probability of errors is high
- Detection when retransmission is easy and probability of errors is low