Big Idea in Data Communications:

A conceptual framework for a data communications system. Multiple sources send to multiple destinations through an underlying physical channel.

Each of the boxes corresponds to one subtopic of data communications:
Signal Encoding Design Goals

- No DC components
- No long sequence of zero-level line signals
- No reduction in data rate
- Error detection ability
- Low cost
Encoding Schemes (Line Coding Mechanisms)

NRZ-L

NRZI

Bipolar-AMI
   (most recent preceding 1 bit has negative voltage)

Pseudoternary
   (most recent preceding 0 bit has negative voltage)

Manchester

Differential Manchester
Nonreturn to Zero-Level (NRZ-L)

- Two different voltages for 0 and 1 bits
- 0 = high level / 1 = low level
NRZI (Nonreturn to Zero – Invert on ones)

- Non-return to zero, **inverted on ones**
- Constant voltage pulse for duration of bit
- Data encoded as presence or absence of signal transition at the beginning of bit time
  - Data is based on **transitions** (low to high or high to low) – level change
  - Where there is a ONE → Transition occurs
  - Where there is a ZERO → No transition occurs

Advantages
- Data represented by changes rather than levels
- **more reliable** detection of transition rather than level – when noise exists!
Multilevel Binary Bipolar-AMI

- AMI stands for alternate mark inversion
- Use **more than two** levels
- Bipolar-AMI
  - zero represented by no line signal
  - one represented by positive or negative pulse
  - One's pulses **alternate** in polarity
  - no loss of sync if a long string of ones
    - long runs of zeros still a problem
  - no net dc component
  - lower bandwidth
  - easy error detection

Bipolar - AMI

0 → 0
1,1 → +,−
Multilevel Binary Pseudoternary

- one represented by absence of line signal
- zero represented by alternating positive and negative
- no advantage or disadvantage over bipolar-AMI
- each used in some applications

1 → 0
0,0 → +,-
Example

- Using NRZI, how do you represent 1 1 1 1 1?
- Assuming it takes 5usec to send 5 bits what is the duration of each bit?
- Assuming it takes 5usec to send 5 bits what is the duration of each signal element?
  - The signal will be 0 1 0 1 0 (toggling – starting with Zero as the initial state)
  - Each bit = 1 usec
  - Each signal element = 1 usec
Scrambling

- The objective is to avoid long sequences of zero level line signals and providing some type of error detection capability
- We compare two techniques:
  - B3ZS (bipolar 8-zero substitution)
  - HDB3 (High-density Bipolar-3 zeros)

B8ZS:
One octet of zero is replaced by:
000VB0VB
V = 1 code violation
Scrambling

- The objective is to avoid long sequences of zero level line signals and providing some type of error detection capability.
- We compare two techniques:
  - B3ZS (bipolar 8-zero substitution)
  - HDB3 (High-density Bipolar-3 zeros)

**HDB3:**
- 4 zeros are replaced by:
  - 000V if the number of pulses (ones) since last substitution was ODD
  - B00V if the number of pulses (ones) since last substitution was EVEN

V = 1 code violation
Channel Coding
Error Correction in SONET

• BIT Interleaved Parity (BIP)
  • Uses Parity Bit
Two Strategies for Handling Channel Errors

• A variety of mathematical techniques have been developed that overcome errors during transmission and increase reliability
  • Known collectively as channel coding

• The techniques can be divided into two broad categories:
  • Forward Error Correction (FEC) mechanisms
  • Automatic Repeat reQuest (ARQ) mechanism

• In either case we are adding overhead
  • There is always a tradeoff - adding redundancy vs. error detection

• What is the impact of channel error?
Error Correction Motivation

- Errors can be detected and corrected
  - Error correction is more complex
- Correction of detected errors usually requires data block to be retransmitted
- Instead need to correct errors on basis of bits received
Error Correction  Basic Idea

• Adds redundancy to transmitted message
• Can deduce original despite some errors
  • Errors are detected using error-detecting code
  • Error-detecting code added by transmitter
  • Error-detecting code are recalculated and checked by receiver

• map $k$ bit input onto an $n$ bit codeword
  • each distinctly different
  • When error occurs the receiver tries to guess which codeword sent was (e.g., teh $\rightarrow$ the)
Error Detection

Detection methods

Parity check

Cyclic redundancy check

Checksum

Error Correction with Row and Column (RAC) Parity
Redundancy Check

1- Vertical Redundancy Check (VRC)  
   - Parity Check
2- Longitudinal Redundancy Check (LRC)
3- Cyclic Redundancy Check
Error Detection – Parity Check

• Basic idea
  • Errors are detected using error-detecting code
  • Error-detecting code added by transmitter
  • Error-detecting code are recalculated and checked by receiver

• Parity bit
  • Odd (odd parity)
    • If it had an even number of ones, the parity bit is set to a one, otherwise it is set to a zero
    • \((P=0 \text{ if odd ones})\rightarrow\) always odd number of ones in the frame
    • Asynchronous applications and Standard in PC memory
  
• Even (even parity)
  • Synchronous applications

F(1110001)→
odd parity 1 111 000 1
Parity Bit + Data Block
Error Detection – Parity Check

An Example Block Error Code:
Single Parity Checking

<table>
<thead>
<tr>
<th>Original Data</th>
<th>Even Parity</th>
<th>Odd Parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0 1 1 0 1 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 1 0 1 0 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 0 0 0 0 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 0 1 0 0 1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

If even number of 1s → Even parity = 0
Error Detection Basic Mechanism

- for block of \( k \) bits transmitter
- Represented by \((n,k)\) encoding scheme
  - \( k \) dataword length
  - \( n \) codeword
  - \( r \) added bits

Example: 8-bit data + single parity bit → \( 2^9 \) (512) possibilities / only \( 2^8 \) (255=256-1) valid code words (excluding all-zero)

What is the minimum number of bits we should add?
Redundancy Check

- **Longitudinal Redundancy Check (LRC)**
  - Organize data into a table and create a parity for each column

Original Data: 11100111 11011101 00111001 10101001

LRC: 10101010

Diagram:

```
  11100111 11011101 00111001 10101001
     |        |        |
     v        v        v
  11011101 00111001 10101001
     |        |        |
     v        v        v
  11100111 10101001
     |        |
     v        v
  10101010
```

Original Data: 11100111 11011101 00111001 10101001 10101010

LRC: 10101010
Hamming Distance: A Measure of a Code's Strength

• No channel coding scheme is ideal!
  • changing enough bits will always transform to a valid codeword

• What is the minimum number of bits of a valid codeword that must be changed to produce another valid codeword?
  • To answer the question, engineers use a measure known as the Hamming distance
  • Given two strings of n bits each, the Hamming distance is defined as the number of differences.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>d(000, 001) = 1</td>
<td>d(000, 101) = 2</td>
</tr>
<tr>
<td>d(101, 100) = 1</td>
<td>d(001, 010) = 2</td>
</tr>
<tr>
<td>d(110, 001) = 3</td>
<td>d(111, 000) = 3</td>
</tr>
</tbody>
</table>
The Tradeoff Between Error Detection and Overhead

• A large value of $d_{\text{min}}$ is desirable
  • because the code is immune to more bit errors, if fewer than $d_{\text{min}}$ bits are changed, the code can detect that error(s) occurred

• The maximum number of bit errors that can be detected:

$$e = d_{\text{min}} - 1$$

• A code with a higher value of $d_{\text{min}}$ sends more redundant information than an error code with a lower value of $d_{\text{min}}$

• Code rate that gives the ratio of a dataword size to the codeword size

$$R = \frac{k}{n}$$
Error Detection and Correction

- Relation between Hamming Distance and Error
  - When a codeword is corrupted during transmission, the Hamming distance between the sent and received codewords is the number of bits affected by the error.

  - Example: if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is \( d(00000, 01101) = 3 \).

- To guarantee the detection of up to \( e \) errors in all cases, the minimum Hamming distance in a block code must be

  \[ d_{\text{min}} = e + 1 \rightarrow e = d_{\text{min}} - 1 \]

- To guarantee the maximum \( t \) correctable errors in all cases

  \[ t = \left\lfloor \frac{d_{\text{min}} - 1}{2} \right\rfloor \]
Cyclic Redundancy Codes (CRC)

- Term **cyclic** is derived from a property of the codewords:
  - A **circular shift** of the bits of any codeword produces another one
- A (n=7, k=4) CRC by Hamming

<table>
<thead>
<tr>
<th>Dataword</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000 000</td>
</tr>
<tr>
<td>0001</td>
<td>0001 011</td>
</tr>
<tr>
<td>0010</td>
<td>0010 110</td>
</tr>
<tr>
<td>0011</td>
<td>0011 101</td>
</tr>
<tr>
<td>0100</td>
<td>0100 111</td>
</tr>
<tr>
<td>0101</td>
<td>0101 100</td>
</tr>
<tr>
<td>0110</td>
<td>0110 001</td>
</tr>
<tr>
<td>0111</td>
<td>0111 010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataword</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000 101</td>
</tr>
<tr>
<td>1001</td>
<td>1001 110</td>
</tr>
<tr>
<td>1010</td>
<td>1010 011</td>
</tr>
<tr>
<td>1011</td>
<td>1011 000</td>
</tr>
<tr>
<td>1100</td>
<td>1100 010</td>
</tr>
<tr>
<td>1101</td>
<td>1101 001</td>
</tr>
<tr>
<td>1110</td>
<td>1110 100</td>
</tr>
<tr>
<td>1111</td>
<td>1111 111</td>
</tr>
</tbody>
</table>
CRC generator and checker

- Example: Division in CRC Encoder

```
Dataword: 1 0 0 1

Quotient:
1 0 1 0

Divisor: 1 0 1 1

0 0 0 0

Remainder: 1 1 0

Codeword: 1 0 0 1 1 1 0
```
**CRC generator and checker**

transmits \( n \) bits which is exactly divisible by some number (**predetermined divisor**)
receiver divides frame by that number

Refer to your notes for examples!
CRC generator and checker

- At the Receiver:
  - Example: Division in CRC Decoder

Codeword: $\begin{array}{c}1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0\end{array}$

Known divisor: $1 \ 0 \ 1 \ 1$

Codeword: $\begin{array}{c}1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0\end{array}$

Division in CRC Decoder

<table>
<thead>
<tr>
<th>Dataword accepted</th>
<th>Syndrome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

Dataword discarded

<table>
<thead>
<tr>
<th>Syndrome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1</td>
</tr>
</tbody>
</table>
Cyclic Redundancy Codes (CRC)
Mathematical Representation

• Let $M(x)$ be the message polynomial
• Let $P(x)$ be the generator polynomial (divisor)
  – $P(x)$ is fixed for a given CRC scheme
  – $P(x)$ is known both by sender and receiver
• Create a block polynomial $F(x)$ based on $M(x)$ and $P(x)$ such that $F(x)$ is divisible by $P(x)$

\[
\frac{F(x)}{P(x)} = Q(x) + \frac{0}{P(x)}
\]
Example of CRC

- **Send**
  - \( M(x) = 110011 \rightarrow x^5 + x^4 + x + 1 \) (6 bits)
  - \( P(x) = 11001 \rightarrow x^4 + x^3 + 1 \) (5 bits, \( n = 4 \))
    \[ \rightarrow 4 \text{ bits of redundancy} \]
  - Form \( x^nM(x) \rightarrow 1100110000 \)
    \[ \rightarrow x^9 + x^8 + x^5 + x^4 \]
  - Divide \( x^nM(x) \) by \( P(x) \) to find \( C(x) \)

\[ \begin{array}{c|c}
100001 & \hline
11001 & 1100110000 \\
11001 & 11001 \\
10000 & \\
11001 & \\
1001 & \rightarrow C(x)
\end{array} \]

Send the block 110011 1001

- **Receive**

\[ \begin{array}{c|c}
11001 & \hline
11001 & 1100111001 \\
11001 & 11001 \\
11001 & 00000 \\
 & \downarrow \text{No remainder} \\
 & \rightarrow \text{Accept}
\end{array} \]

\[ \frac{F(x)}{P(x)} = Q(x) + \frac{0}{P(x)} \]
Forward Error Correction

- Used in OTN (10Gbps)
- RS codes:
  - $n$ (symbols) = $k$ (symbols) + $r$ (symbols) $\rightarrow$ 125 usec
  - 1 symbol has $m$ bits
  - $2^m - 1 = n$ symbols
- Example:
  - $N = 255$; $r = 16; k = 239$
  - Each symbol is 8 bytes
- Uses Reed-Solomon codes
  - (255,239), $r = 16$; 7 (16/239) percent redundancy, Corrected errors: $r/2 = 8$
  - (255,223), $r = 16$; 15 (32/223) percent redundancy, Corrected errors: $r/2 = 16$
• Assume $n=4$, $k=2 \rightarrow$ Code rate $\frac{1}{2}$

• Given BER, coding can improve $Eb/No$
  - Lower $Eb/No$ is required
  - **Code gain** is the reduction in dB in $Eb/No$ for a given BER
    - E.g., for BER=$10^{-6} \rightarrow$ code gain is 2.77 dB

• Energy per coded bit ($Eb$) = $\frac{1}{2}$ data bit ($E_b$)
  - Hence, BER will be 3dB less
  - This is because $E_{bit}=2\times E_{data}$

• For very high BER, adding coding requires **higher** $Eb$
  - Not due to overhead