# Data Link Layer

## Error Control: Introduction and Detection

Geoff Merrett

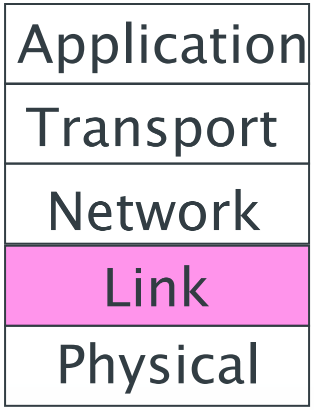
ELEC3227/ELEC6255: Networks

*See Tanenbaum Chapter 3 (Data Link Layer)*

Image: http://i.ytimg.com/vi/DULSCrW9_Ss/maxresdefault.jpg

# Outline

- Error Control
  - Introduction
  - Error Detection
    - Parity (Single, Multiple, Interleaved)
    - Checksums
    - Cyclic Redundancy Checks (CRCs)
  - Error Correction
    - Hamming Codes



# Introduction

- Some transmission media are reasonably error free
  - e.g. a fibre optic link
- Some are not
  - e.g. a wireless link

- Error models
  - Single-bit errors
    - e.g. an extreme value of thermal noise
  - Bursts of errors
    - e.g. deep fading in a wireless channel;
    - e.g. transient electrical interference on a wired link etc

# Introduction

- We can use two strategies:
  - Error detection (using error detection codes), and then re-request frame
  - Error correction (using error correcting codes), aka Forward Error Correction (FEC)

- We should use something appropriate
  - If few errors, FEC will introduce unnecessary overhead
  - If many errors, retransmissions will introduce unnecessary overhead

- Might also be handled in other layers
  - FEC at the Physical and some higher layers (e.g. in real-time content streaming)
  - Error detection in Data Link, Network and Transport layers

# Redundancy

- We embed an ability to detect and/or correct errors by adding redundancy

$$m \text{ (data bits)} + r \text{ (redundant check bits)} = n \text{ (total number of transmitted bits)}$$

- This is then known as an $(n, m)$ code with an $n$-bit codeword

- Only a fraction of the $2^n$ possible codewords are used: $2^m/2^n = 2^{-r}$
  - It is this sparseness or redundancy that allows errors to be detected/corrected.

- The code rate is the fraction of the codeword that carries non-redundant information, i.e. = $m/n$
  - Noisy channel, a code rate of 0.5 might be suitable;
  - Reliable channel, a code rate close to 1.0 might be suitable.

# Block, systematic and linear codes

- Block code
  - Any set of input bits (the $m$-bit message) could be looked up in a table to find the corresponding $n$-bit codeword
    - i.e. the $r$ check bits are solely a function of the $m$-bit message

- Systematic code
  - The set of input bits (the $m$-bit message) is transmitted alongside the $r$ check bits
    - i.e. the $m$-bits in the message appear in the $n$-bits in the codeword

- Linear code
  - The check bits are a linear function of the message bits

# Hamming distance

- Hamming distance $d$ = the number of bits different between two codewords
  - XOR and count the 1's

- If you know the algorithm used to create codewords, you can construct the complete set of all possible codewords.
  - min($d$) is the Hamming distance of the complete code

- If two codewords are a Hamming distance $d$ apart, it will require $d$ single-bit errors to convert one into another

- Block codes can detect $d - 1$ errors and correct $(d - 1)/2$ errors

# Error Detection

# Single Parity Bit

- Even parity
  - add a redundant parity bit to make an even number of 1's
  - Equivalent of an XOR operation
  - E.g. 1110000 → 11100001

- Odd parity
  - add a redundant parity bit to make an odd number of 1's

- To check, see whether an even (or odd) number of bits are received

# Single Parity Bit

- If a single-bit error occurs:
  - e.g. 11100101; detected, sum is wrong
  - To be expected, as Hamming distance $d = 1$.

- If a double-bit error occurs:
  - e.g. 2 errors, 11101101; *not detected*, sum is correct!

- If any odd number of bit-errors occur:
  - e.g. 3 errors, 11011001; detected sum is wrong

- Can detect any odd number of errors (including in the parity bit itself)
  - i.e. random errors are detected with probability 0.5!

# Multiple Parity Bits

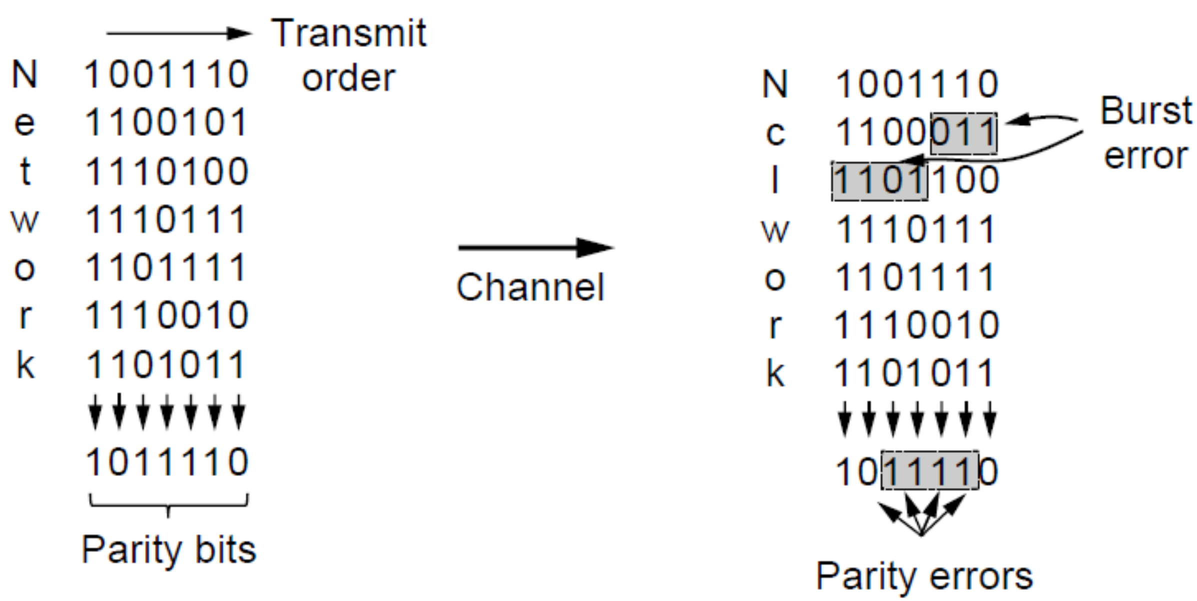
- Consider the message as a matrix $w$ bits wide and $h$ bits high
  - Send a separate parity bit for each $w$-bit row
  - $r$ (where $r = h$) bit errors can be detected (provided max of 1 error occurs per row)
  - We add $r$ parity bits e.g. for the message: 101100011000 (with $w = 3$, $h = 4$, and even parity):

| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |

- Still not much good for burst errors (as maximum of 1 error per row!)

# Interleaved Parity

- Interleaving is a general technique
  - Converts a code that detects (or corrects) isolated errors…
  - …into one that detects (or corrects) burst errors
- Interleaving $r$ (where $r = w$) parity bits detects burst errors up to length $r$
  - Calculate parity bits over columns, but send data along rows
  - Parity bits are sent at the end
  - Each parity sum is made over non-adjacent bits
  - An even burst of up to $r$ errors will not cause it to fail



# Checksums

- Often just used to refer to a set of check bits (e.g. parity bits)
- However, stronger checksums are based on summing the message bits
  - and are usually placed at the end of the message
- Typically treat the message as being formed from many $N$-bit words, and adds $N$ (i.e. $r = N$) check bits to the end which are the modulo $2^N$ sum of all words
  - e.g. Internet 16-bit 1's complement checksum
- More effective than parity
  - e.g. if the LSB in two words have single bit flips, parity would not detect.
  - Detects bursts of up to $r = N$ errors, and random errors with probability $1\text{-}2^N$
  - Vulnerable to systematic errors, e.g., added zeros, swapping parts of the message etc
- Other methods of creating a checksum can provide stronger protection
  - e.g. Fletcher's checksum: improves protection against changes in the position of data

# Cyclic Redundancy Checks (CRCs)

- Parity and checksums are rarely used in the Data Link Layer
  - A stronger error detection code is in widespread use: the CRC or polynomial code
  - e.g. can detect all double bit errors and not vulnerable to systematic errors

- Treats bit strings as the coefficients of a polynomial
  - A $k$-bit string is regarded as the coefficient of a polynomial with $k$ terms
  - i.e. $x^{k-1} + x^{k-2} + \ldots + x^{1} + x^{0}$ (a polynomial with a 'degree' = $k$-1)

- The protocol has an agreed generator polynomial, $G(x)$
  - The degree of $M(x)$ (the polynomial corresponding to the $m$-bit frame) must be greater than that of $G(x)$

# Calculating CRCs

- Adds bits so that transmitted frame viewed as a polynomial is evenly divisible by a generator polynomial

Start by adding $r$ (= $k$-1) 0's to RHS of the $M(x)$ bit string (the frame) – so that it becomes $x^r \cdot M(x)$

Divide the bit string corresponding to $G(x)$ into $x^r \cdot M(x)$ using modulo 2 division

Create the $n$-bit bit string corresponding to $T(x)$ for transmission by subtracting the remainder from $x^r \cdot M(x)$ using modulo-2 subtraction

This makes $T(x)$ evenly divisible

```
            Frame:  1 1 0 1 0 1 1 1 1 1
        Generator:  1 0 0 1 1

                    1 1 0 0 0 0 1 1 1 0   ← Quotient (thrown away)
1 0 0 1 1 ) 1 1 0 1 0 1 1 1 1 1 0 0 0 0   ← Frame with four zeros appended
            1 0 0 1 1
              1 0 0 1 1
              1 0 0 1 1
                0 0 0 0 1
                0 0 0 0 0
                  0 0 0 1 1
                  0 0 0 0 0
                    0 0 1 1 1
                    0 0 0 0 0
                      0 1 1 1 1
                      0 0 0 0 0
                        1 1 1 1 0
                        1 0 0 1 1
                          1 1 0 1 0
                          1 0 0 1 1
                            1 0 0 1 0
                            1 0 0 1 1
                              0 0 0 1 0
                              0 0 0 0 0
                                    1 0   ← Remainder

Transmitted frame:  1 1 0 1 0 1 1 1 1 1 0 0 1 0   ← Frame with four zeros appended
                                                    minus remainder
```

# Detecting Errors using CRCs

- $T(x)$ (the transmitted $n$-bit message) should clearly always divide exactly by $G(x)$ as the remainder has been subtracted from it, e.g.
  - 123 / 10 = 12 remainder 3
  - (123 – 3) / 10 = 12 remainder 0
- On decoding, all the receiver has to do is calculate $T(x) / G(x)$.
  - If the result has a remainder = 0, no errors were detected.
  - If errors have occurred during transmission, i.e. $T(x) + E(x)$, this will give a non-zero remainder when $(T(x) + E(x)) / G(x)$ is calculated.
  - The remainder will only be zero if $E(x)$ is a factor of $G(x)$
    - *See book for more information on performance*
- CRCs are easy to calculate/check in hardware using shift and XOR operations

# IEEE 802 and CRC-32

- IEEE 802.3 (Ethernet) uses CRC-32
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + 1$
  - Detects:
    - all burst errors of length 32 and less
    - all bursts affecting an odd number of bits

- IEEE 802.15.4 and LoRaWAN use the CCITT (ITU-T) CRC-16
  - $x^{16} + x^{12} + x^{5} + 1$

# CRCs and Parity

- Even parity is the same as CRC-1
  - $G(x) = 1\,x^{1} + 1\,x^{0} = (x + 1)$
  - e.g. apply to the message: $0110101_2$

Electronics and
Computer Science

UNIVERSITY OF
Southampton

# Error Correction

# Hamming Codes

- Consider a code that will allow all single-bit errors to be corrected
- Only a fraction of the $2^n$ bit patterns (possible codewords) are used: $2^m/2^n = 2^{-r}$
  - The others can be considered 'illegal' bit patterns
- Each of the $2^m$ messages must have $n$ illegal bit patterns one bit away from it
  - Identified by systematically inverting each of the $n$-bits in the valid codeword
- Therefore, each of the $2^m$ messages requires $n + 1$ bit patterns dedicated to it
  - Since there are a total of $2^n$ bit patterns, $(n + 1) \cdot 2^m \leq 2^n$
  - Because $n = m + r$, we can say that $(m + r + 1) \leq 2^r$
  - This gives a theoretical lower-limit on $r$, the number of check bits needed to correct single errors
  - This theoretical limit can be achieved using Hamming codes

# Hamming Codes

- The $n$ data bits are consecutively numbered from bit 1 at the left
- Bits in positions that are powers of 2 (1, 2, 4, 8, etc) are the $r$ check bits
- The remaining bits are sequentially filled up with the $m$ message bits

- The check bits are just even (or odd) parity bits
- Each of the $m$ message bits may contribute to multiple check bits
  - Rewrite the location of a message bit as its powers of two, and those are the positions it contributes to

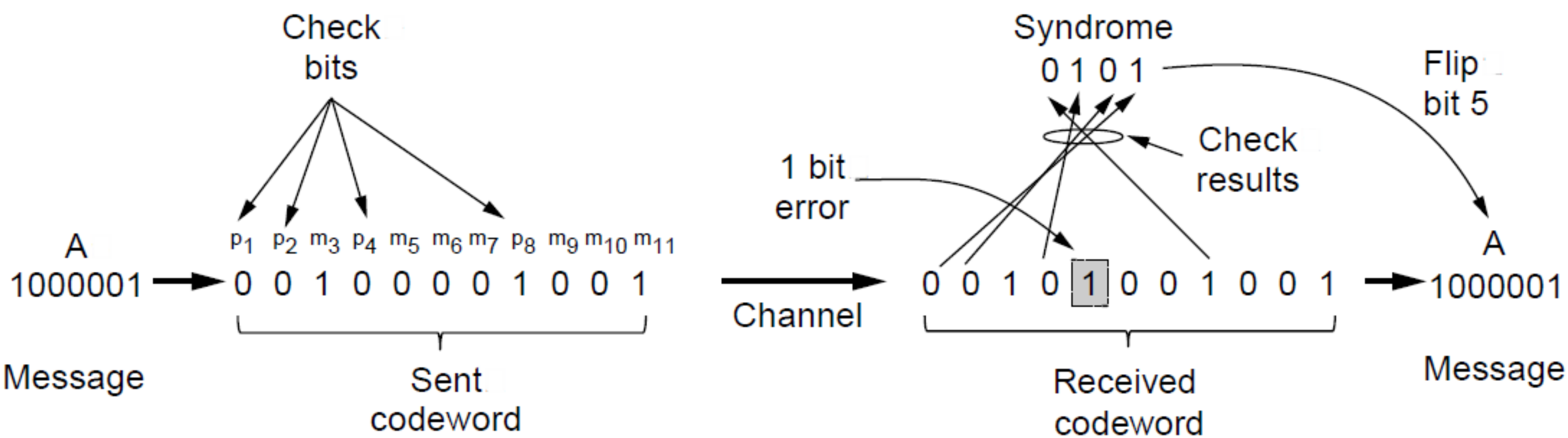
- Hamming codes provide a code with a Hamming distance $d = 3$
  - An $m$=7-bit message, resulting in an $n$=11-bit codeword, is an (11, 7) Hamming code

# Hamming Codes

- Detecting errors
  - To ‘deconstruct’ the Hamming code, the check bits are recalculated at the destination (including the check-bit) – these are the *check results*
  - If even parity was used, the parity sum should be 0
  - If not, an error has been detected

- Correcting errors
  - If an error has been detected, the set of *check results* become the *error syndrome*
  - This indicates which bit was erroneous, and hence can simply be flipped

# Hamming Codes - example




(11, 7) Hamming code adds 4 check bits and can correct 1 error

# Hamming Codes vs Parity

- Consider a channel with a BER of $10^{-6}$ and a block size of 1000 bits
- To correct a single-bit error (Hamming code):
  - We know that $(m + r + 1) \leq 2^r$
  - Therefore, $1001 \leq 2^r - r$, and hence we'd need to add 10 check bits to each block
  - Therefore, a Mb of data would require an overhead of 10,000 check bits
- To detect a single-bit error (Parity)
  - We need only 1 check-bit per block
  - Therefore, a Mb of data would require an overhead of 1000 check bits
- Once every 1000 blocks, an error will occur
  - Error correction corrects the error: i.e. total overhead = 10,000 bits
  - Error detection retransmits the block: i.e total overhead = 1000 + 1001 = 2001 bits
- Therefore, in this case, error detection has a fifth of the overhead!

# ELEC3222 18/19 Exam Question

A 16-bit packet, $1101\ 1111\ 1010\ 0110_2$, is passed from the network layer to the data link layer for transmission.

1. The data link layer first applies interleaved even parity, with a width of 4 bits. **Calculate** the bit string produced.

2. **State** how many errors this can detect and correct. **Compare** this in terms of error detection/correction and overhead when compared to non-interleaved even parity (i.e. a parity bit after every 4 bits)?

3. The data link layer then frames the data using nibble (4-bit) stuffing. The flag and escape nibbles are $0110_2$ and $1111_2$ respectively. Using your answer from (1), **calculate** the bit string produced. If you were unable to answer (1), assume the bit string was unchanged.

# ELEC3227 19/20 Exam Question

A CRC is calculated for the bit string $10011101_2$, appended to the end, and transmitted. The generator $G(x) = x^3 + 1$.

1. **Calculate** the actual transmitted bit string.

2. The most-significant-bit (MSB) is inverted during transmission. **Show** whether the receiver detects this and, if so, **explain** the steps it might take to rectify it.

3. **State** an example of a bit error in the transmitted bit string that will not be detected, and **explain** why.

# Questions?