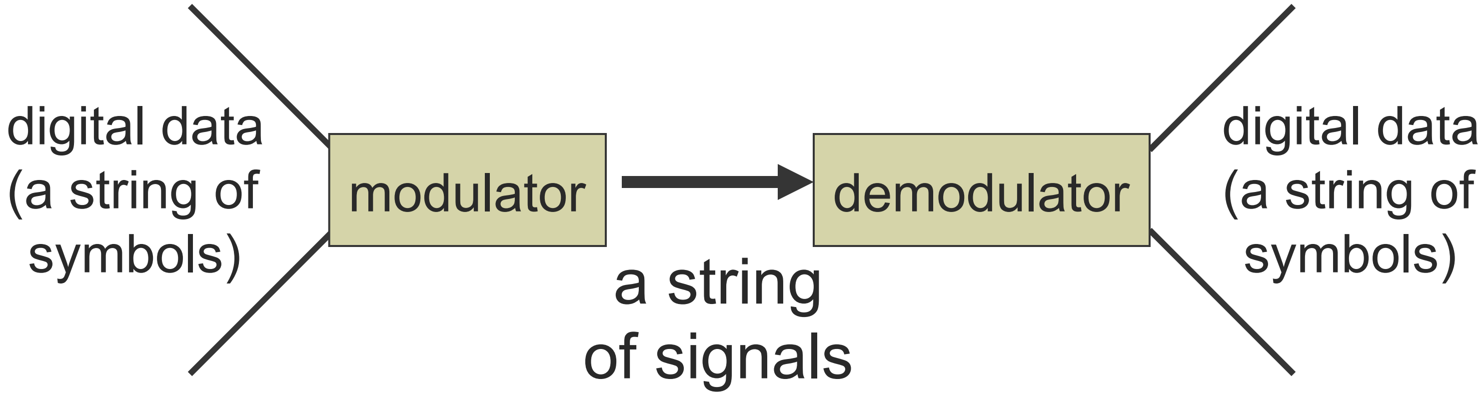
Direct Link Networks – Error Detection and Correction

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

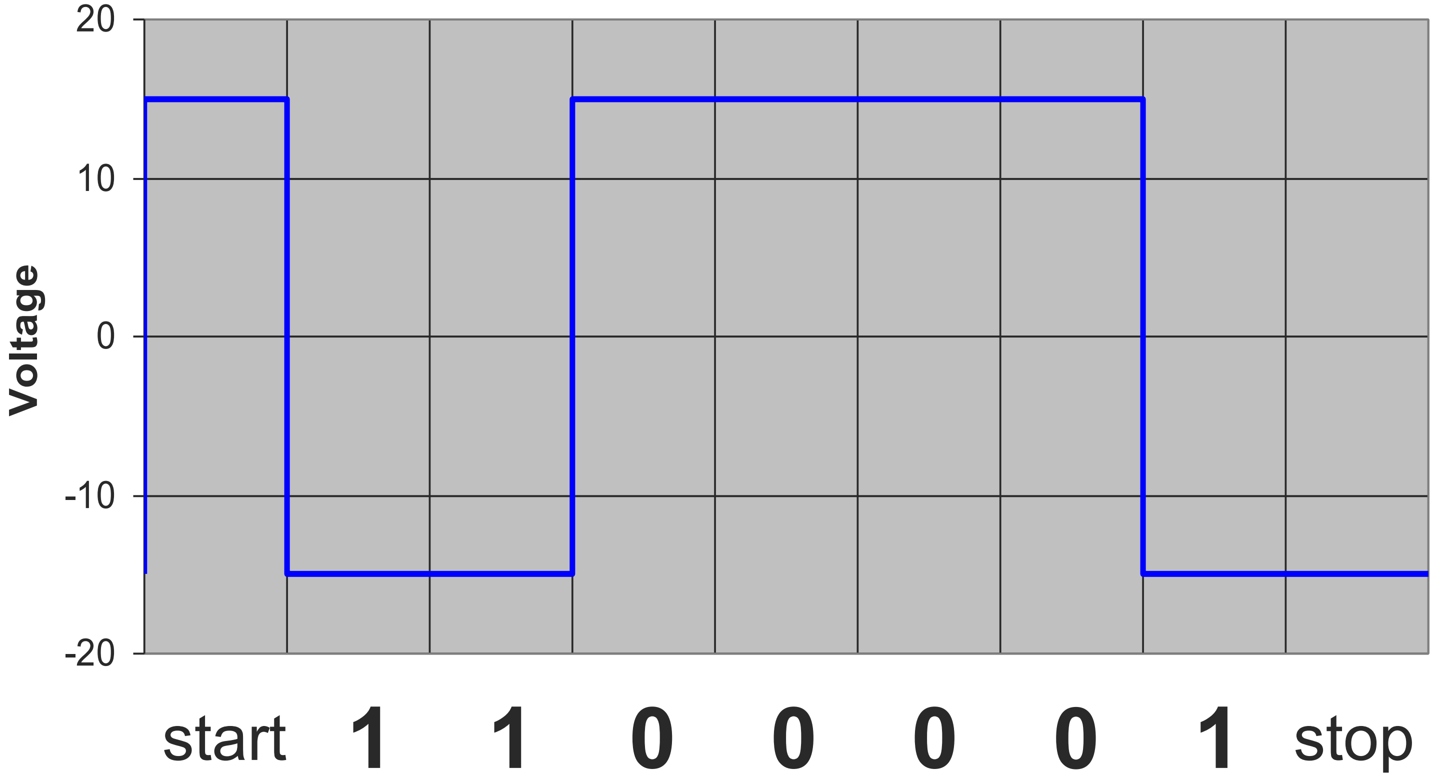
Error Detection

- 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

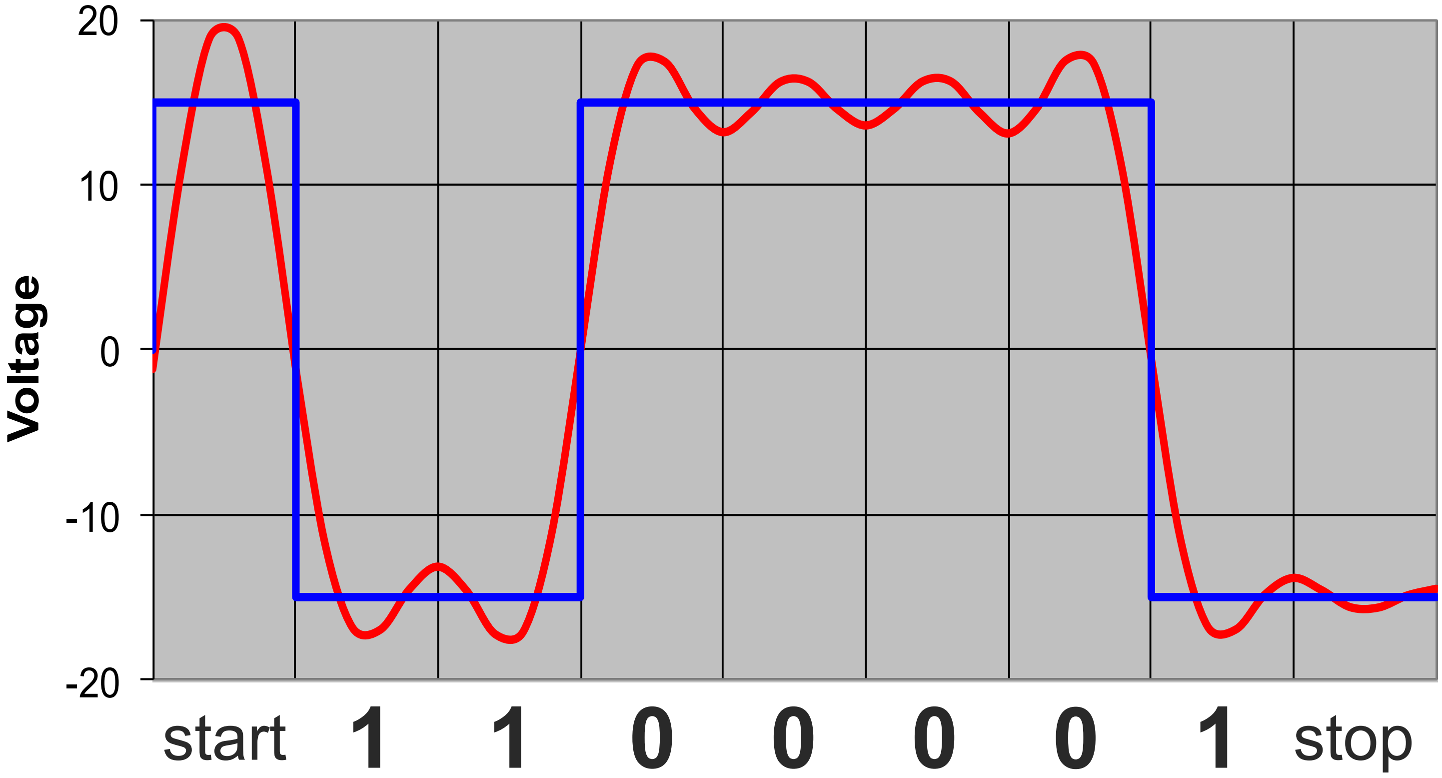
- Consider RS-232 encoding of character ‘Q’
- Assume idle wire (-15V) before and after signal

RS-232 Encoding of 'Q'



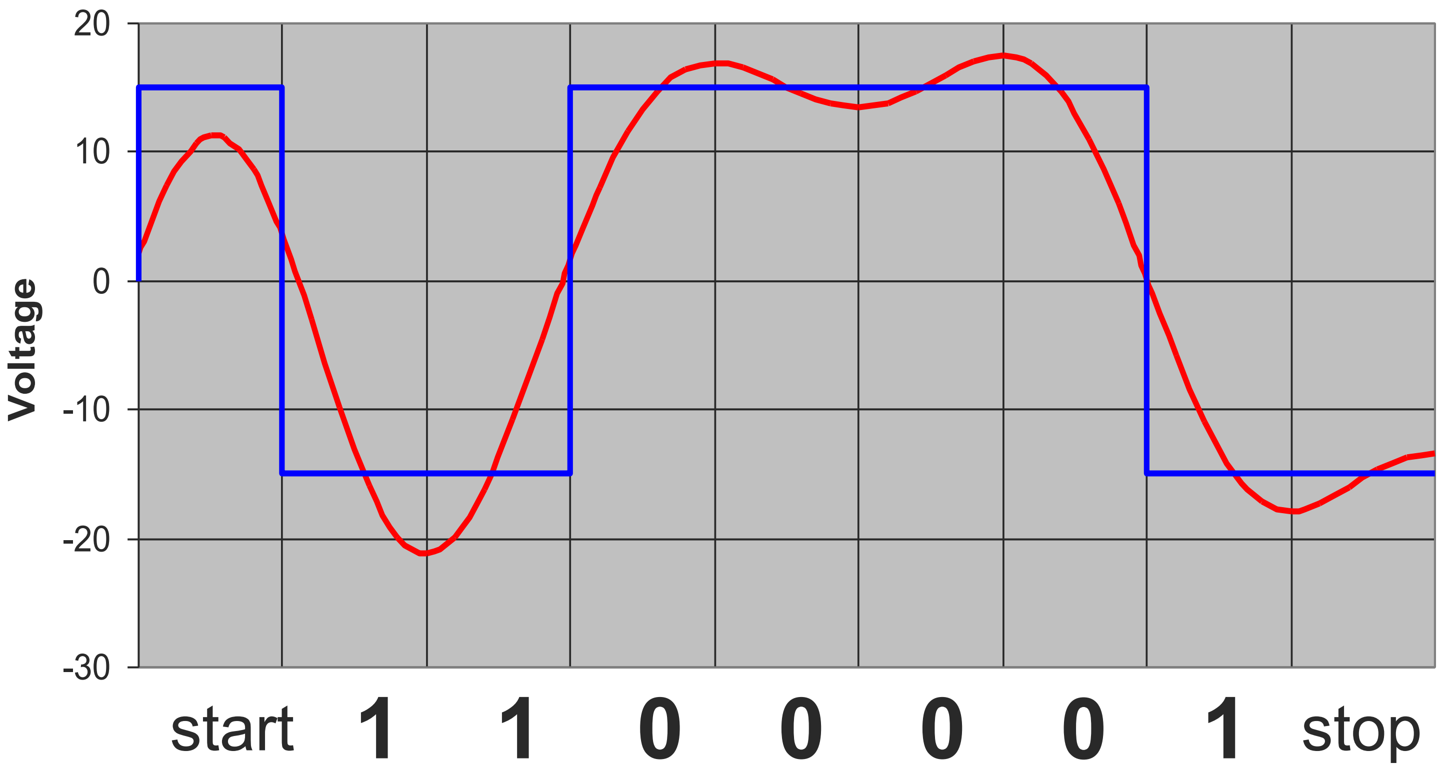
Encoding isn't perfect

Example with bandwidth = baud rate

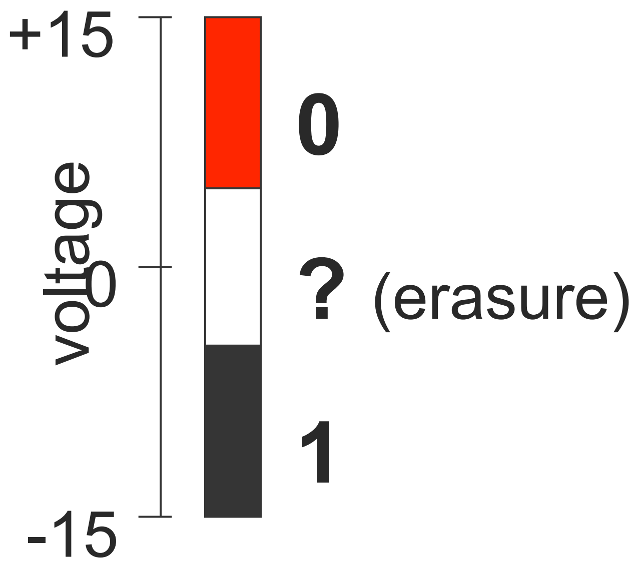


Encoding isn't perfect

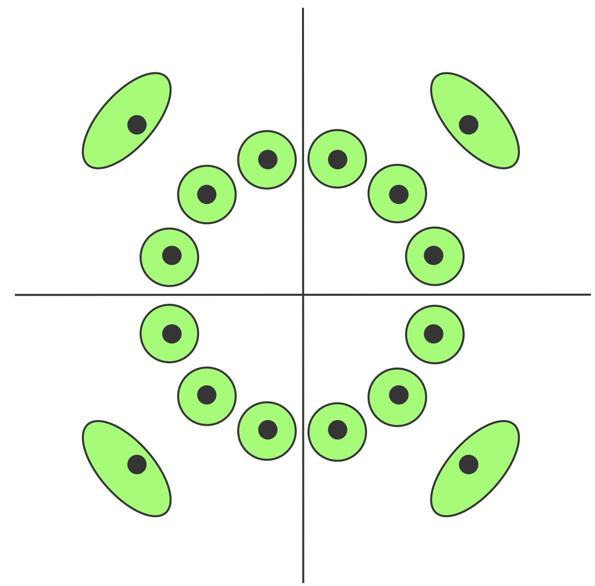
Example with bandwidth = baud rate/2



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

Single bit error

No bit errors

$$P_1 = (1 - P_b)^F$$

Undetected errors

$$P_2 = 1 - P_1$$

Detected errors

$$P_3 = 0$$

- F = Number of bits per frame

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

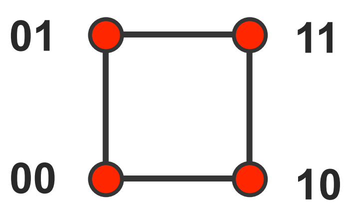
Parity

- 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

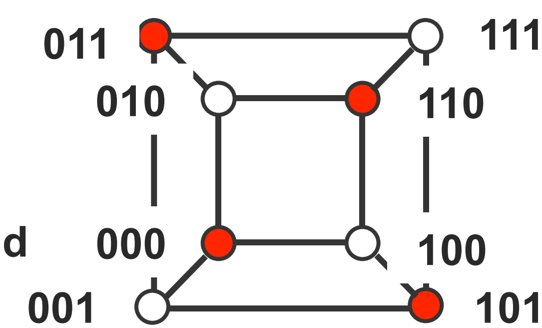
- 1-bit error detection with parity
  - Add an extra bit to a code to ensure an even (odd) number of 1s
  - Every code word has an even (odd) number of 1s

Valid code words



Parity Encoding:

White – invalid (error)



Voting: Correcting Bit Flips

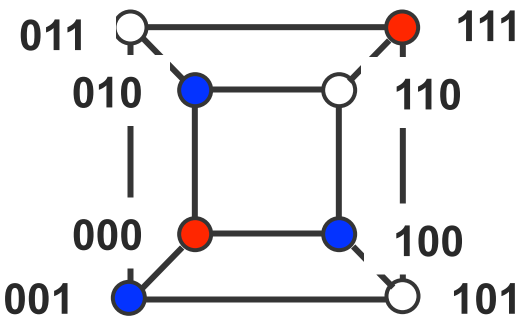
- 1-bit error correction with voting
  - Every codeword is transmitted n times
  - Codeword is 3 bits long

Valid code words

Voting:

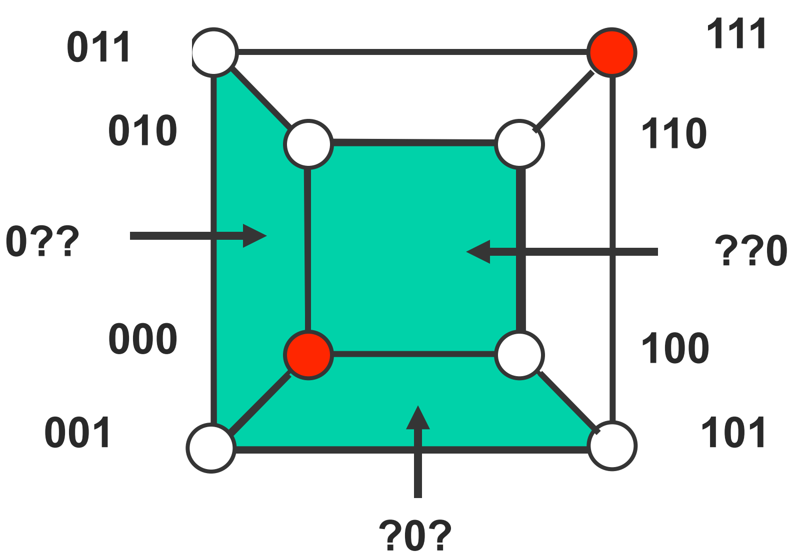
White – correct to 1

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

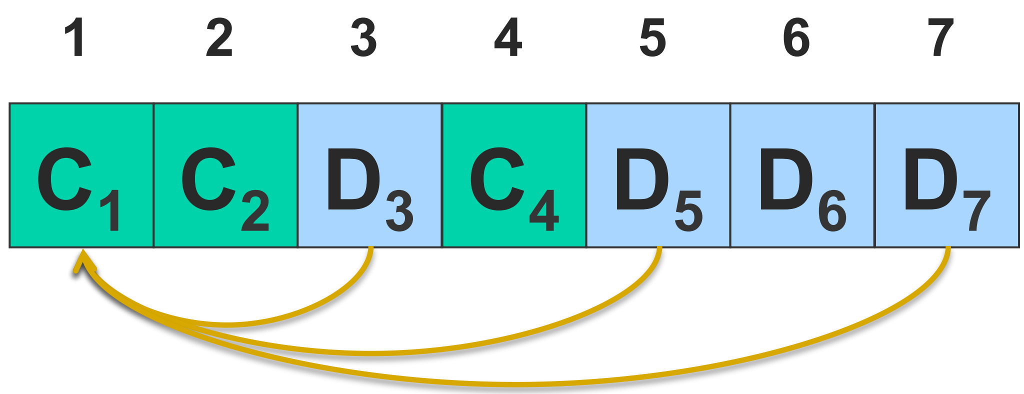
- N-bit error detection
  - No code word changed into another code word
  - Requires Hamming distance of N+1
- N-bit error correction
  - N-bit neighborhood: all codewords within N bit flips
  - No overlap between N-bit neighborhoods
  - Requires hamming distance of 2N+1

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

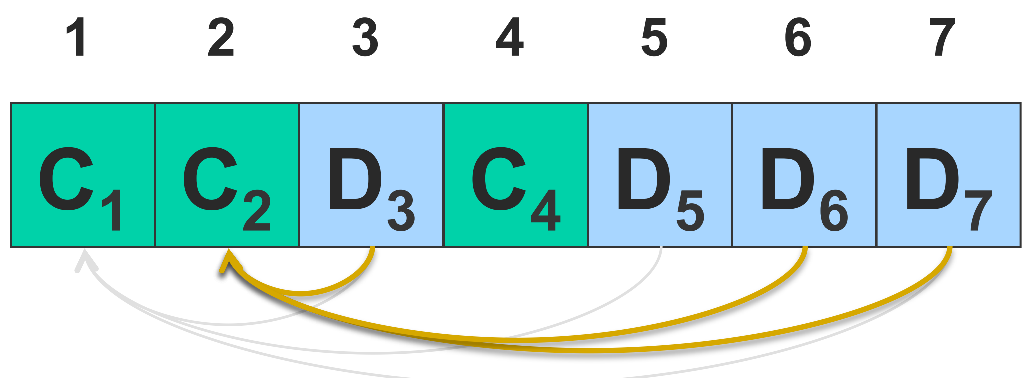
Hamming Codes

- 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* AND *k*) = *j*
- Example:
  - 4 bits of data, 3 check bits



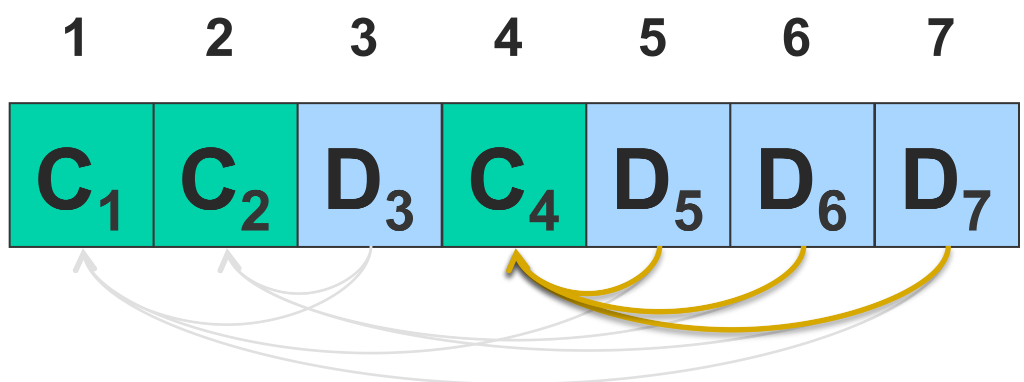
Hamming Codes

- 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

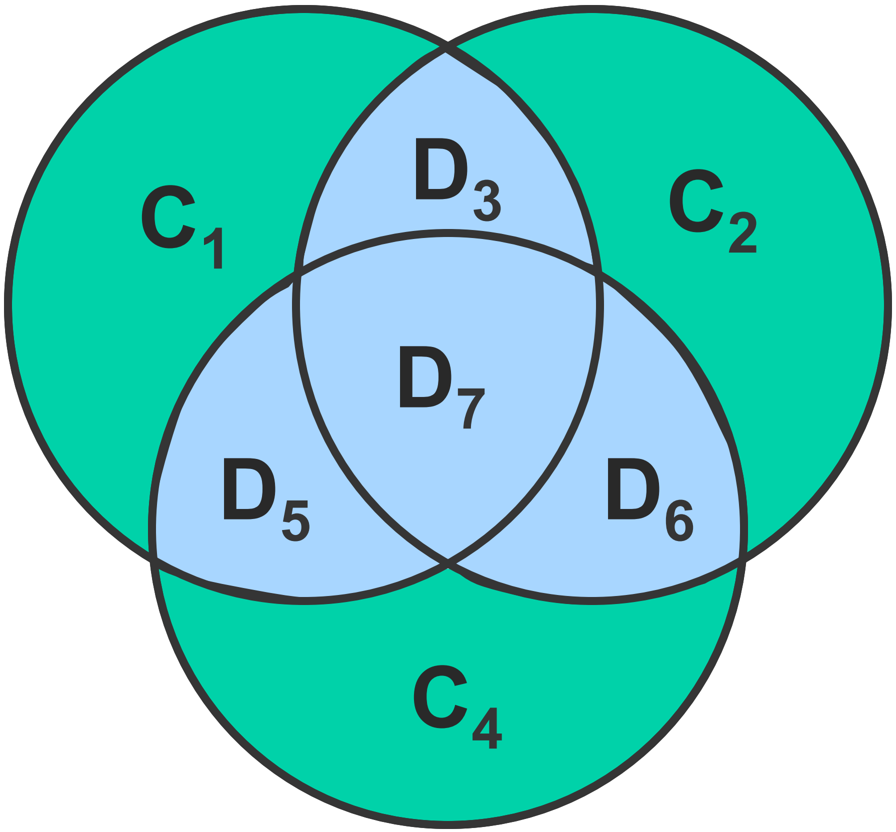


Hamming Codes

- 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



Hamming Codes



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

		Parity Bits
Data	0101001	1
	1101001	0
	1011110	1
	0001110	1
	0110100	1
	1011111	0
Parity Byte	1111011	0

- 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

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

What about 4-bit errors?

Are there any 4-bit errors this scheme *can* detect?

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

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

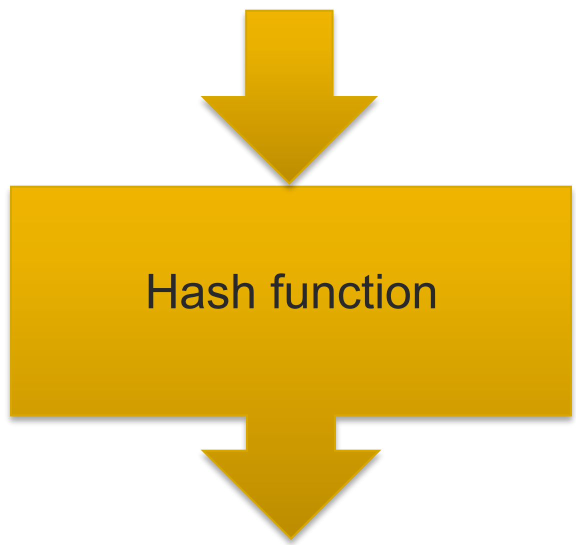
IP Checksum

```
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);
}
```

What could cause this check to fail?

Main Goal: Check the Data!

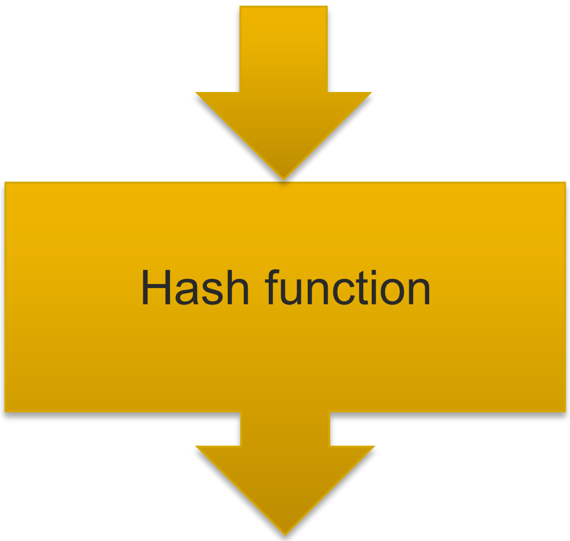
n data bits



k pseudorandom
check bits

Main Goal: Check the Data!

n data bits



k pseudorandom check bits

- 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 $M(x)$

$$= \mathbf{1} *x^7 + \mathbf{0} *x^6 + \mathbf{0} *x^5 + \mathbf{1} *x^4 + \mathbf{1} *x^3 + \mathbf{0} *x^2 + \mathbf{1} *x + \mathbf{0}$$
$$= x^7 + x^4 + x^3 + x$$

- 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 $C(x)$
  - $P(x) = M(x) * x^k + k$ check bits

How can we determine these k bits?

Properties of Polynomial Arithmetic

- Coefficients are modulo 2

  $(x^3 + x) + (x^2 + x + 1) = \ldots$

  $\ldots x^3 + x^2 + 1$

  $(x^3 + x) - (x^2 + x + 1) = \ldots$

  $\ldots x^3 + x^2 + 1$ 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

CRC - Sender

- Given
  - $M(x) =$ 10011010 $= x^7 + x^4 + x^3 + x$
  - $C(x) =$ 1101 $= x^3 + x^2 + 1$
- Steps
  - $T(x) = M(x) * x^k$ (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	Generator
$M(x) =$	$x^7 + x^4 + x^3 + x$	= 10011010	Message

```
1101 | 10011010000    <- Message plus k zeros
        1101
        ----
         1001
         1101
         ----
          1000
          1101
          ----
           1011
           1101
           ----
            1100
            1101
            ----
             1000
             1101
             ----
              101
```

k + 1 bit check sequence c, equivalent to a degree-k polynomial

Remainder

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

```
1101 | 10011010101
       1101
       ----
        1001
        1101
        ----
         1000
         1101
         ----
          1011
          1101
          ----
           1100
           1101
           ----
             1101
             1101
             ----
                0
```

k + 1 bit check sequence c, equivalent to a degree-k polynomial

Received message, no errors

Remainder

m mod c

Result:

CRC test is passed

CRC – Example Decoding – with Errors

$C(x) = x^3 + x^2 + 1 = 1101$ Generator

$P(x) = x^{10} + x^7 + x^5 + x^4 + x^2 + 1 = 10010110101$ Received Message

```
1101 | 10010110101
        1101
        ----
         1000
         1101
         ----
          1011
          1101
          ----
           1101
           1101
           ----
            0101
```

Received message

Two bit errors

k + 1 bit check sequence c, equivalent to a degree-k polynomial

Remainder

m mod c

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 $x^4 + x^2 + x$

```
      1011
     10110
     ------
      1101
     1101
    1101
    -------
 00011111110
```

This is a multiple of c, so that if errors occur according to this sequence, the CRC test would be passed

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 ≥ k

Common Polynomials for C(x)

CRC	C(x)
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

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