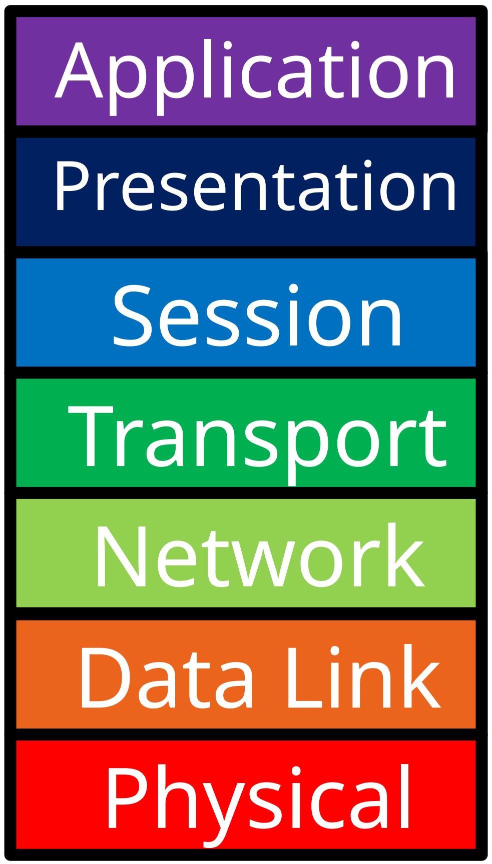
Computer Networks

Lecture 2: Data Link

Based on slides from D. Choffnes Northeastern U. and P. Gill from StonyBrook University
Revised Autumn 2015 by S. Laki

Data Link Layer



- Function:
  - Send blocks of data (frames) between physical devices
  - Regulate access to the physical media
- Key challenge:
  - How to delineate frames?
  - How to detect errors?
  - How to perform media access control (MAC)?
  - How to recover from and avoid collisions?

Outline

- Framing
- Error Checking and Reliability
- Media Access Control
  - 802.3 Ethernet
  - 802.11 Wifi

Framing

- Physical layer determines how bits are encoded
- Next step, how to encode blocks of data
  - Packet switched networks
  - Each packet includes routing information
  - Data boundaries must be known so headers can be read
- Types of framing
  - Byte oriented protocols
  - Bit oriented protocols
  - Clock based protocols

Byte Oriented: Byte Stuffing

- Add **FLAG** bytes as sentinel to the beginning and end of the data
- Problem: what if **FLAG** appears in the data?
  - Add a special **DLE** (Data Link Escape) character before **FLAG**
  - What if **DLE** appears in the data? Add **DLE** before it.
  - Similar to escape sequences in C
    - printf(“You must \”escape\” quotes in strings”);
    - printf(“You must \\escape\\ forward slashes as well”);
- Used by Point-to-Point protocol, e.g. modem, DSL, cellular

Byte Oriented: Byte Counting

- Sender: insert length of the data in bytes at the beginning of each frame
- Receiver: extract the length and read that many bytes
- What happens if there is an error transmitting the count field?

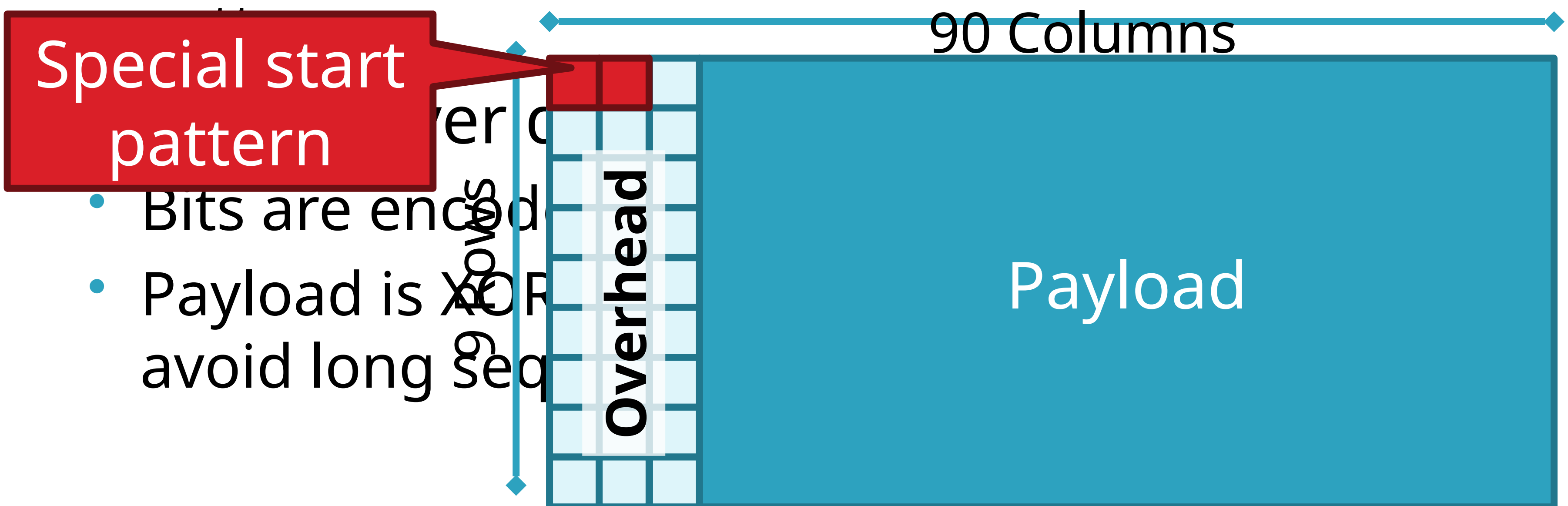
Bit Oriented: Bit Stuffing

01111110	Data	01111110

- Add sentinels to the start and end of data (similarly to byte stuffing)
  - Both sentinels are the same
  - Example: 01111110 in High-level Data Link Protocol (HDLC)
- Sender: insert a 0 after each 11111 in data
  - Known as “bit stuffing”
- Receiver: after seeing 11111 in the data…
  - 11111**0** → remove the 0 (it was stuffed)
  - 11111**1** → look at one more bit
    - 11111**10** → end of frame
    - 11111**11** → error! Discard the frame
- Disadvantage: 20% overhead at worst
- What happens if error in sentinel transmission?

Clock-based Framing: SONET

- **S**ynchronous **O**ptical **Net**work
  - Transmission over very fast optical links
  - STS-*n*, e.g. STS-1: 51.84 Mbps, STS-768: 36.7 Gbps
- STS-1 frames based on fixed sized frames
  - 9*90 = 810 bytes after 810 bytes look for start
  - ...ver c...
  - Bits are encod...
  - Payload is XOR...
    avoid long seq...



Outline

- Framing
- Error Checking
- Media Access Control
  - 802.3 Ethernet
  - 802.11 Wifi

Dealing with Noise

- The physical world is inherently noisy
  - Interference from electrical cables
  - Cross-talk from radio transmissions, microwave ovens
  - Solar storms
- How to detect bit-errors in transmissions?
- How to recover from errors?

Naïve Error Detection

- Idea: send two copies of each frame
  - if (memcmp(frame1, frame2) != 0) { OH NOES, AN ERROR! }
- Why is this a bad idea?
  - Extremely high overhead
  - Poor protection against errors
    - Twice the data means twice the chance for bit errors

Parity Bits

- Idea: add extra bits to keep the number of 1s even
  - Example: 7-bit ASCII characters + 1 parity bit

0101001 1 1101001 0 1011110 1 0001110 1 0110100 1

10

- Detects 1-bit errors and some 2-bit errors
- Not reliable against bursty errors

Error control

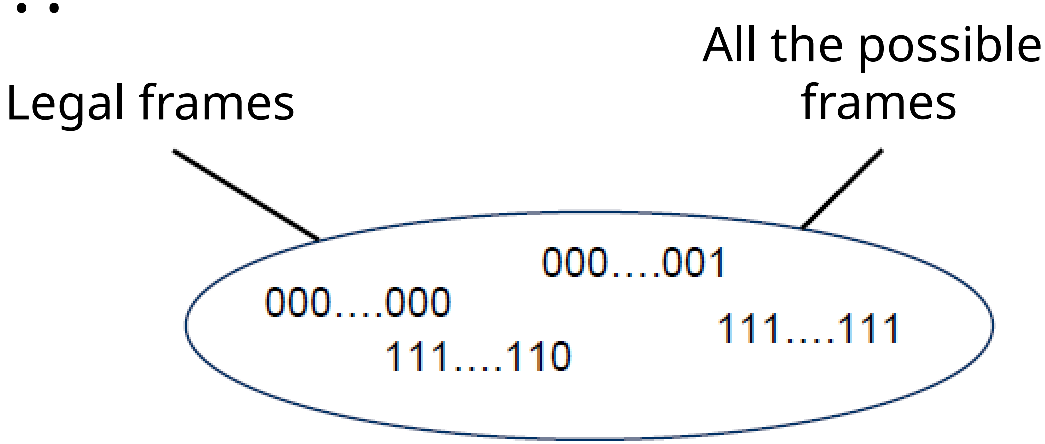
- Error Control Strategies
  - Error Correcting codes (Forward Error Correction (FEC))
  - Error detection and retransmission Automatic Repeat Request (ARQ)

Error control

- Objectives
  - Error detection
    - with correction
      - Forward error correction
    - without correction -> e.g. drop a frame
      - Backward error correction
      - The erroneous frame needs to be retransmitted
  - Error correction
    - without error detection
      - e.g. in voice transmission

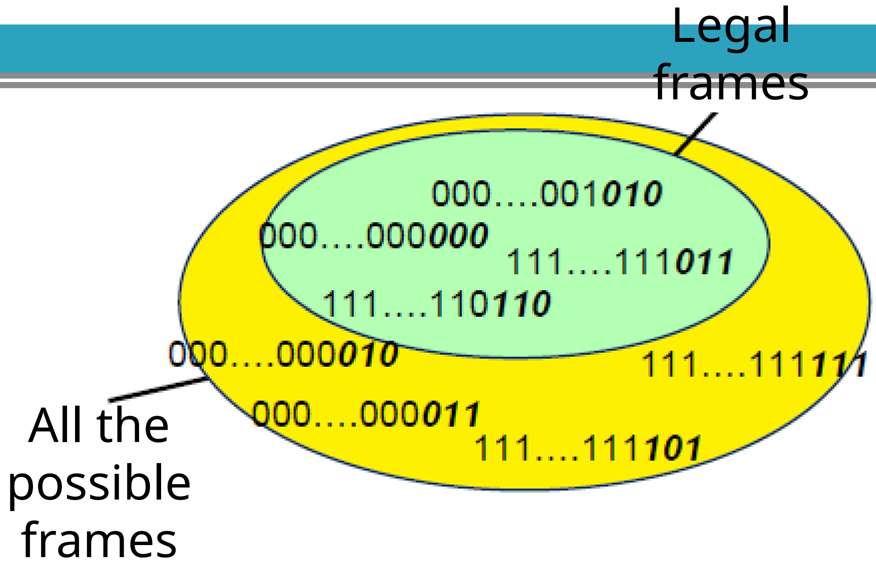
Redundancy

- Redundancy is required for error control
- Without redundancy
  - 2^m possible data messages can be represented as data on m bits
  - They all are legal!!!
  - Each error results a new legal data message
- How to detect errors???



Error-correcting codes
Redundancy

- A frame consists of
  - m data bits (message)
  - r redundant/check bits
  - The total length n = m + r

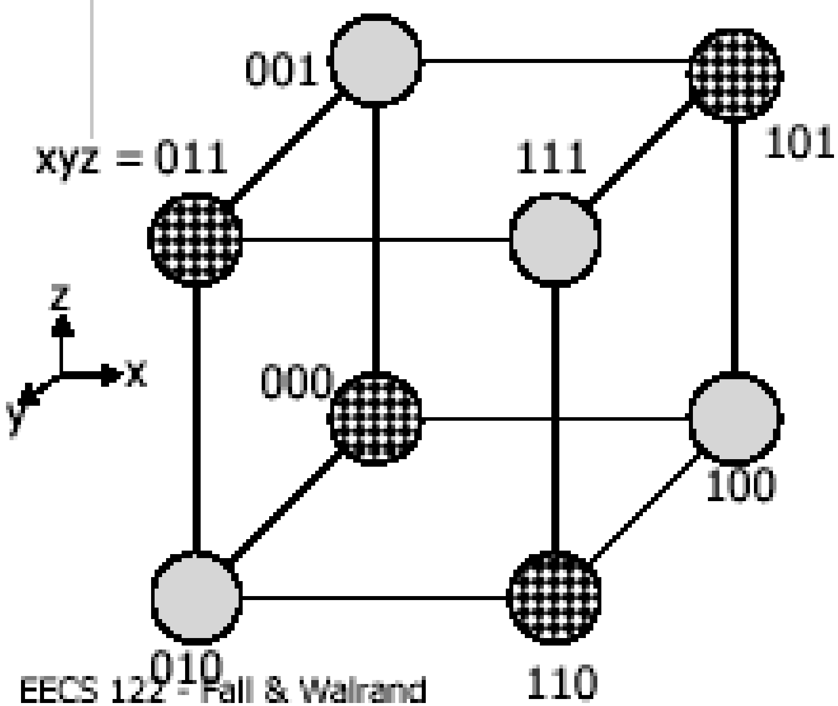


- This n-bit unit is referred to as an n-bit codeword!

Error Control Codes

How Codes Work: Words and Codewords

- Code = subset of possible words: Codewords
- Example:
  - 3 bits => 8 words; codewords: subset



Words:
000, 001, 010, 011
100, 101, 110, 111

Code:
000, 011, 101, 110

Send only codewords

Hamming distance

- The Hamming distance between two codewords is the number of differences between corresponding bits.

1. The Hamming distance d(000, 011) is 2 because

$$\boxed{000 \oplus 011 \text{ is } 011 \text{ (two 1s)}}$$

2. The Hamming distance d(10101, 11110) is 3 because

$$\boxed{10101 \oplus 11110 \text{ is } 01011 \text{ (three 1s)}}$$

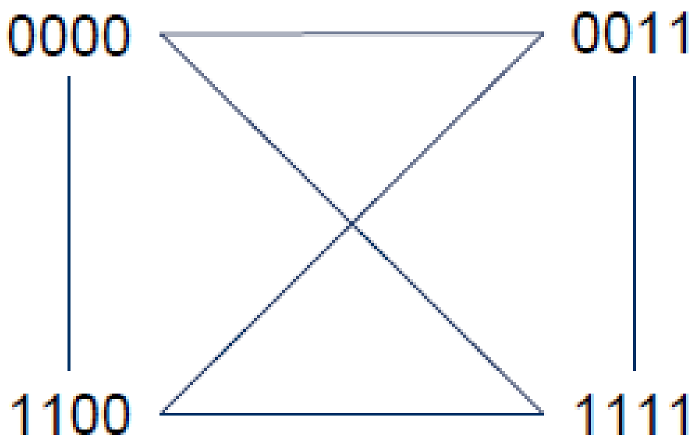
Hamming distance

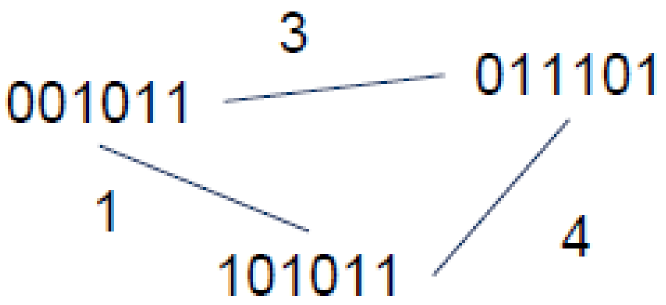
- If not all the 2^n possible codewords are used
  - Set of legal codewords =: S
- Hamming distance of the complete code
  - The smallest Hamming distance of between all the possible pairs in the set of legal codewords (S)

$$d(S) = \min_{x,y \in S, x \neq y} d(x,y)$$

What is the Hamming distance?

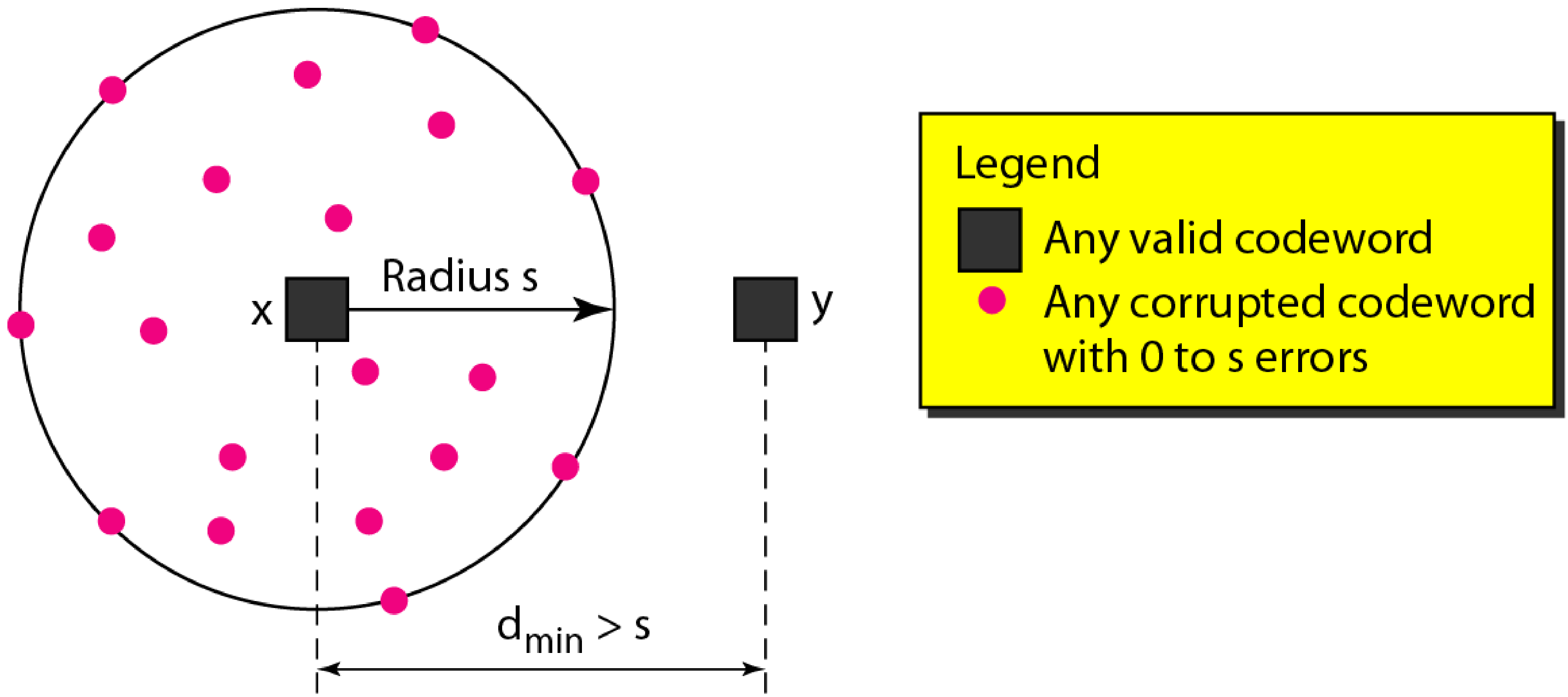
- Two examples:





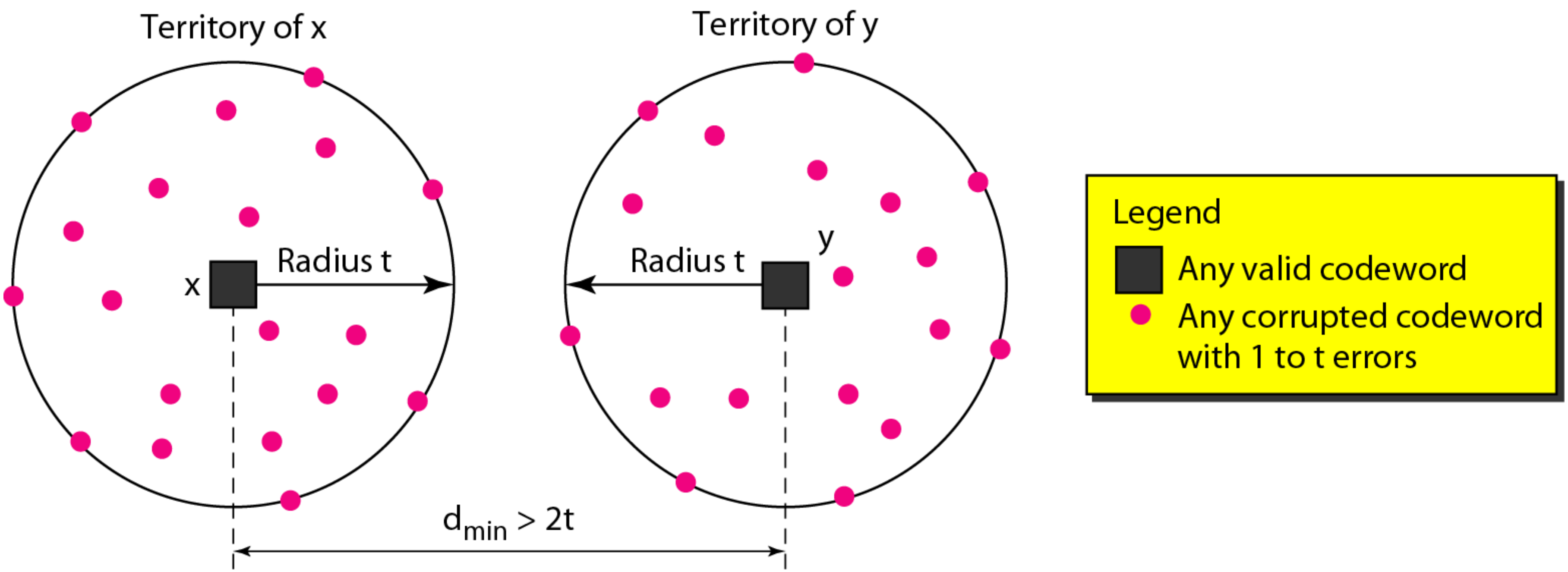
Error detection

To detect d errors, you need a distance d+1 code.



Error correction

To correct d errors, you need a distance 2d+1 code.



Example

S={ 00000000,

00001111,

11110000,

11111111

}

Parity bit – already discussed

- A single parity bit is appended to the data
  - Choosen according to the number of 1 bits in the message
    - odd or even

- An example using even parity
  - Original message: 1011010
  - A 0 bit is added to the end: 10110100
  - m=8 and r=1 in this case

- The distance of this code is 2, since any single-bit error produces a codeword with the wrong parity.

Checksums

- Idea:
  - Add up the bytes in the data
  - Include the sum in the frame

START	Data	Checksum	END

- Use ones-complement arithmetic
- Lower overhead than parity: 16 bits per frame
- But, not resilient to errors
  - Why? 1 101001 + 0 101001= 10010010
- Used in UDP, TCP, and IP

Cyclic Redundancy Check (CRC)

- Uses field theory to compute a semi-unique value for a given message

- Much better performance than previous approaches
  - Fixed size overhead per frame (usually 32-bits)
  - Quick to implement in hardware
  - Only 1 in 2^{32} chance of missing an error with 32-bit CRC

CRC (Cyclic Redundancy Check)

- Polynomial code
  - Treating bit strings as representations of polynomials with coefficients of 0 and 1.
- CRC
  - Add k bits of redundant data to an n-bit message.
  - Represent n-bit message as an n-1 degree polynomial;
    - e.g., MSG=10011010 corresponds to $M(x) = x^7 + x^4 + x^3 + x^1$.
  - Let k be the degree of some divisor polynomial G(x);
    - e.g., $G(x) = x^3 + x^2 + 1$.
    - Generator polynomial
      - Agreed upon it in advance

CRC

- Transmit polynomial P(x) that is evenly divisible by G(x), and receive polynomial P(x) + E(x);
  - E(x)=0 implies no errors.

- Recipient divides (P(x) + E(x)) by G(x);
  - the remainder will be zero in only two cases:
    - E(x) was zero (i.e. there was no error),
    - or E(x) is exactly divisible by C(x).

- Choose G(x) to make second case extremely rare.

A basic example with numbers

- Make all legal messages divisible by 3
- If you want to send 10
  - First multiply by 4 to get 40
  - Now add 2 to make it divisible by 3 = 42
- When the data is received ..
  - Divide by 3, if there is no remainder there is no error
  - If no error, divide by 4 to get sent message
- If we receive 43, 44, 41, 40, then error
- 45 would not be recognized as an error

Mod 2 arithmetic

- Operations are done modulo 2

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A - B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

```
  0110111011
+ 1101010110
= 1011101101
```

```
 1111
+1010
=====
 0101
```

```
     11001
     x 101
    =====
     11001
+  11001
=========
   1111101
```

A basic example with polynomials

- Sender:
  - multiply *M(x)* = $x^7 + x^4 + x^3 + x^1$ by x^k; for our example, we get
    - $x^{10} + x^7 + x^6 + x^4$ (10011010000);
  - divide result by *C(x)* (1101);

```
                              11111001
Generator   1101    | 10011010000        Message
                      1101
                       1001
                       1101
                        1000
                        1101
                         1011
                         1101
                          1100
                          1101
                            1000
                            1101
                             101         Remainder
```

Send 10011010000 + 101 = 10011010101,
since this must be exactly divisible by *C(x)*;

Further properties

- Want to ensure that G*(x)* does not divide evenly into polynomial *E(x)*.
- All single-bit errors, as long as the x^k and x^0 terms have non-zero coefficients.
- All double-bit errors, as long as *G(x)* has a factor with at least three terms.
- Any odd number of errors, as long as *G(x)* contains the factor *(x + 1)*.
- Any "burst" error (i.e sequence of consecutive errored bits) for which the length of the burst is less than *k* bits.
- Most burst errors of larger than *k* bits can also be detected.

Even Parity

Actually consists of using x+1 polynomial
Given message 0111, multiply by x to get 01110
Now divide by x+1=11

```
     0101
   ______
11 | 01110
      11
      0010
         11
          1=remainder
```

Message = 01110+1=01111 even parity

- 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	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$