18-345 – Fall 08

Lecture 6
Digital Transmission Fundamentals

Peter Steenkiste

reading: Chapter 3

Outline Physical Layer Lectures

- Digital representation of information
  – Digital representation of analog signals
- Analog versus digital transmission
- Basic properties of dig. transmission
- Fundamental limits of dig. transmission
- Line coding, modulation
  – Amplitude, frequency, and phase modulation
- Properties of transmission media
- Synchronization
- Error detection and error correction

Announcements

- Quiz today!
- Project 1 due on Thursday
- Powering optical amplifiers
  – Powering using DC from both ends – powered devices in series
  – Also diagnostic capabilities: monitoring, switching to spare fiber, etc.
  – Can look at equipment specs

Error Detection and Correction

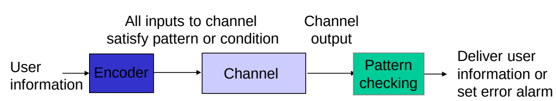
Reading: Section 3.9

Error Control

- Channels introduce errors in digital communications
- Applications require certain reliability level
  – Data applications require error-free transfer
  – Voice & video applications tolerate some errors
- Error control may be needed to meet application requirement
- Error control ensures a data stream is transmitted to a certain level of accuracy despite errors
- Two basic approaches:
  – Error ***detection*** & retransmission (ARQ)
  – Forward error ***correction*** (FEC)

Key Idea

- All transmitted data blocks (“codewords”) are chosen so that they satisfy a pattern
- If received block doesn’t satisfy pattern, it is in error
- Redundancy: Only a subset of all possible blocks can be valid codewords
- Undetectable Error: When channel transforms a codeword into another valid codeword



Single Parity Check

- Append an parity bit to *k* information bits

Info Bits: $b_1, b_2, b_3, \ldots, b_k$

Check Bit: $b_{k+1} = b_1 + b_2 + b_3 + \ldots + b_k$ modulo 2

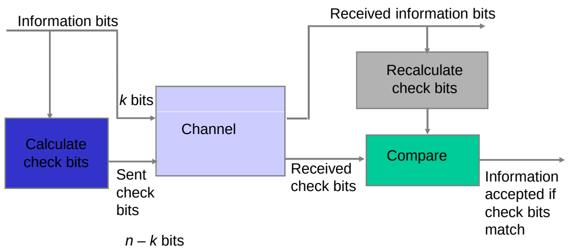
Codeword: $(b_1, b_2, b_3, \ldots, b_k, b_{k+1})$

- All codewords have even # of 1s
- Receiver checks to see if # of 1s is even
  - All error patterns that create an odd # of 1 bits are detectable
  - All even-numbered error patterns are undetectable
- ASCII code is precisely such as code (7+1 bits)

Example of Single Parity Code

- Information (7 bits): (0, 1, 0, 1, 1, 0, 0)
- Parity Bit: $b_8 = 0 + 1 + 0 + 1 + 1 + 0 = 1$
- Codeword (8 bits): (0, 1, 0, 1, 1, 0, 0, 1)
- If single error in bit 3 : (0, 1, 1, 1, 1, 0, 0, 1)
  - # of 1's =5, odd
  - Error detected
- If errors in bits 3 and 5: (0, 1, 1, 1, 0, 0, 0, 1)
  - # of 1's =4, even
  - Error not detected

Parity Checkbits & Error Detection



How good is the single parity check code?

- *Redundancy*: Single parity check code adds 1 redundant bit per k information bits: overhead = 1/(k+1)
- *Coverage*: all error patterns with odd # of errors can be detected
  - An error pattern is a binary (k+1)-tuple with 1's where errors occur and 0's elsewhere
  - Of 2^{k+1} binary (k+1)-tuples, ½ are odd, so 50% of error patterns can be detected
- Is it possible to detect more errors if we add more check bits?
- Yes, with the right codes

What if bit errors are random?

- Many transmission channels introduce bit errors at random, independently of each other, and with probability *p*
- Some error patterns are more probable than others:

$$P[10000000] = p(1-p)^7 = (1-p)^8\left(\frac{p}{1-p}\right) \text{ and}$$

$$P[11000000] = p^2(1-p)^6 = (1-p)^8\left(\frac{p}{1-p}\right)^2$$

- In any worthwhile channel $p<0.5$, and so $(p/(1-p))<1$
- It follows that patterns with 1 error are more likely that patterns with 2 errors and so forth
- What is the probability that an undetectable error pattern occurs?

Single parity check code with random bit errors

- Undetectable error pattern if even # of bit errors:

$$P[\text{error detection failure}] = P[\text{undetectable error pattern}]$$
$$= P[\text{error patterns with even number of 1s}]$$
$$= \binom{n}{2} p^2 (1-p)^{n-2} + \binom{n}{4} p^4 (1-p)^{n-4} + \ldots$$

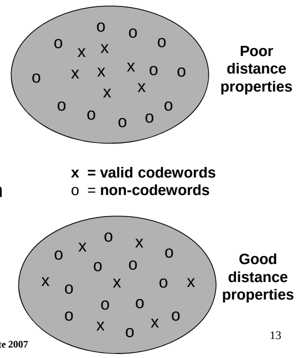
- Example: Evaluate above for n=32, $p=10^{-3}$

$$P[\text{undetectable error}] = \binom{32}{2}(10^{-3})^2(1-10^{-3})^{30} + \binom{32}{4}(10^{-3})^4(1-10^{-3})^{28}$$
$$\approx 496\,(10^{-6}) + 35960(10^{-12}) \approx 4.96(10^{-4})$$

- For this example, roughly 1 in 2000 transmissions will result in an undetectable error

What is a good code?

- Most channels will have relatively few bit errors
- Erroneous codewords transmitted over those channels will map to nearby n-tuples
- If valid codewords are close to each other, then detection failures may occur
- Good codes should maximize separation between valid codewords



Two-Dimensional Parity Check

- More parity bits to improve coverage
- Arrange information as columns
- Add single parity bit to each column
- Add a final "parity" column
- Used in early error control systems

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

Last column consists of check bits for each row

Bottom row consists of check bit for each column

Error-detecting capability

One error:

```
1 0 0 1 0|0
0 0 0 0 0|1  <-
1 0 0 1 0|0
1 1 0 1 1|0
---------
1 0 0 1 1|1
  ^
```

Two errors:

```
1 0 0 1 0|0
0 0 0 0 0|1  <-
1 0 0 1 0|0
1 0 0 1 1|0  <-
---------
1 0 0 1 1|1
```

Three errors:

```
1 0 0 1 0|0
0 0 0 1 0|1
1 0 0 1 0|0
1 0 0 1 1|0  <-
---------
1 0 0 1 1|1
      ^
```

Four errors:

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

1, 2, or 3 errors can always be detected; Not all patterns >4 errors can be detected

Arrows indicate failed check bits

Other Error Detection Codes

- Many applications require very low error rate
- Need codes that detect more number of errors
- Single parity check codes do not detect enough errors
- Two-dimensional codes require too many check bits
- The following error detecting codes are widely used in practice:
  - Internet Check Sums
  - CRC Polynomial Codes

Internet Checksum

- Several Internet protocols (e.g. IP, TCP, UDP) use check bits to detect errors in the ***header***
- A checksum is calculated for header contents and included in a special field.
- Checksum is potentially recalculated at every router, so algorithm selected for ease of implementation in software
- Let header consist of L, 16-bit words,

  $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{L-1}$
- The algorithm appends a 16-bit checksum $\mathbf{b}_L$

Checksum Calculation

The checksum $\mathbf{b}_L$ is calculated as follows:

- Treating each 16-bit word as an integer, find

  $\mathbf{x} = \mathbf{b}_0 + \mathbf{b}_1 + \mathbf{b}_2 + \ldots + \mathbf{b}_{L-1}$ modulo $2^{16}-1$
- The checksum is then given by:

  $\mathbf{b}_L = -\mathbf{x}$ modulo $2^{16}-1$

Thus, the headers must satisfy the following ***pattern*** at the receiver:

$\mathbf{0} = \mathbf{b}_0 + \mathbf{b}_1 + \mathbf{b}_2 + \ldots + \mathbf{b}_{L-1} + \mathbf{b}_L$ modulo $2^{16}-1$

- The checksum calculation is carried out in software using one's complement arithmetic

Internet Checksum Example

Use Modulo Arithmetic

- Assume 4-bit words
- Use mod 2^4-1 (= 15) arithmetic
- b_0=1100 = 12
- b_1=1010 = 10
- b_0+b_1=12+10=7 mod15
- b_2 = -7 = 8 mod15
- Therefore
- b_2=1000

Use Binary Arithmetic

- Note 16 =1 mod15
- So: 10000 = 0001 mod15
- leading bit wraps around

b_0 + b_1 = 1100+1010
=10110
=10000+0110
=0001+0110
=0111
=7
Take 1's complement
b_2 = -0111 =1000

Polynomial Codes

- Polynomials instead of vectors for codewords
- Polynomial arithmetic instead of check sums
- Implemented using shift-register circuits
- Also called *cyclic redundancy check (CRC)*
- Most data communications standards use polynomial codes for error detection
  - Have very simple hardware implementations
- Polynomial codes also basis for powerful error-correction methods

Binary Polynomial Arithmetic

- Binary vectors map to polynomials

$$(i_{k-1}, i_{k-2}, \ldots, i_2, i_1, i_0) \rightarrow i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_2x^2 + i_1x^1 + i_0$$

Addition:

$$\begin{aligned}(x^7 + x^6 + 1) + (x^6 + x^5) &= x^7 + x^6 + x^6 + x^5 + 1 \\ &= x^7 + (1+1)x^6 + x^5 + 1 \\ &= x^7 + x^5 + 1 \quad \text{since} \quad 1+1 = 0 \bmod 2\end{aligned}$$

Multiplication:

$$\begin{aligned}(x+1)(x^2+x+1) &= x(x^2+x+1) + 1(x^2+x+1) \\ &= (x^3+x^2+x) + (x^2+x+1) \\ &= x^3 + 1\end{aligned}$$

Binary Polynomial Division

- Division with Decimal Numbers

```
        34     <- quotient
 35 ) 1222     <- dividend
       105
divisor 172
       140
        32     <- remainder
```

dividend = quotient x divisor + remainder

1222 = 34 x 35 + 32

- Polynomial Division

```
                 x^3 + x^2 + x          = q(x) quotient
 x^3 + x + 1 ) x^6 + x^5                  <- dividend
 divisor       x^6 +       x^4 + x^3
                     x^5 + x^4 + x^3
                     x^5 +       x^3 + x^2
                           x^4 +       x^2
                           x^4 +       x^2 + x
                                             x  = r(x) remainder
```

Note: *Degree of r(x) is less than degree of divisor*

Polynomial Coding

- k information bits define polynomial of degree k-1

$$i(x) = i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_2x^2 + i_1x + i_0$$

- Code has binary *generating polynomial* of degree n-k

$$g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \ldots + g_2x^2 + g_1x + 1$$

- Find *remainder polynomial* of at most degree n-k-1

```
          q(x)
g(x) ) x^(n-k) i(x)          x^(n-k) i(x) = q(x)g(x) + r(x)
          r(x)
```

- Define the *codeword polynomial* of degree n-1

$b(x) = x^{n-k}i(x) + r(x)$

n bits = k bits + n-k bits

Polynomial example: k=4, n-k=3

Generator polynomial: *g(x)*= $x^3 + x + 1$

Information: (1,1,0,0) *i(x)* = $x^3 + x^2$

Encoding: $x^3 i(x) = x^6 + x^5$

```
                x^3 + x^2 + x                          1110
x^3 + x + 1 ) x^6 + x^5                      1011 ) 1100000
              x^6 +       x^4 + x^3                 1011
                    x^5 + x^4 + x^3                  1110
                    x^5 +       x^3 + x^2            1011
                          x^4 +       x^2             1010
                          x^4 +       x^2 + x         1011
                                            x          010
```

Transmitted codeword:
b(x) = $x^6 + x^5 + x$
b = (1,1,0,0,0,1,0)

The *Pattern* in Polynomial Coding

- All codewords satisfy the following **pattern**:

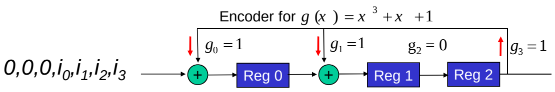
$b(x) = x^{n-k}i(x) + r(x) = q(x)g(x) + r(x) + r(x) = q(x)g(x)$

- All codewords are a multiple of *g(x)!*
- Receiver should divide received n-tuple by g(x) and check if remainder is zero
- If remainder is non-zero, then received n-tuple is not a codeword

Shift-Register Implementation

1. Accept information bits $i_{k-1}, i_{k-2}, \ldots, i_2, i_1, i_0$
2. Append n-k zeros to information bits
3. Feed sequence to shift-register circuit that performs polynomial division
4. After n shifts, the shift register contains the remainder

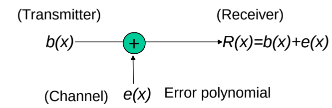
Feedback-Shift Register Circuit



Clock	Input	Reg 0	Reg 1	Reg 2
0	-	0	0	0
1	$1 = i_3$	1	0	0
2	$1 = i_2$	1	1	0
3	$0 = i_1$	0	1	1
4	$0 = i_0$	1	1	1
5	0	1	0	1
6	0	1	0	0
7	0	**0**	**1**	**0**
	Check bits:	$r_0 = 0$	$r_1 = 1$	$r_2 = 0$

$\Longrightarrow r(x) = x$

Undetectable error patterns



- *e(x)* has 1's in error locations & 0's elsewhere
- Receiver divides the received polynomial *R(x)* by *g(x)*
- Undetectable error: If *e(x)* is a multiple of *g(x)*, that is, *e(x)* is a non-zero codeword, then
  $R(x) = b(x) + e(x) = q(x)g(x) + q'(x)g(x)$
- *The set of undetectable error polynomials is the set of nonzero code polynomials*
- *Choose the generator polynomial so that selected error patterns can be detected.*

Designing good polynomial codes

- Select generator polynomial so that likely error patterns are not multiples of *g(x)*
- *Detecting Single Errors*
  - $e(x) = x^i$ for error in location $i+1$
  - If *g(x)* has more than 1 term, it cannot divide x^i
- *Detecting Double Errors*
  - $e(x) = x^i + x^j = x^i(x^{j-i}+1)$ *where* $j>i$
  - If *g(x)* has more than 1 term, it cannot divide x^i
  - If g(x) is a *primitive* polynomial, it cannot divide x^m+1 for all $m<2^{n-k}-1$ (Need to keep codeword length less than $2^{n-k}-1$)
  - Primitive polynomials can be found by consulting coding theory books

Standard Generator Polynomials

CRC = cyclic redundancy check

- CRC-8: $= x^8 + x^2 + x + 1$ ATM
- CRC-16: $= x^{16} + x^{15} + x^2 + 1$
  $= (x+1)(x^{15} + x + 1)$ Bisync
- CCITT-16: $= x^{16} + x^{12} + x^5 + 1$ HDLC, XMODEM, V.41
- CCITT-32: IEEE 802, DoD, V.42
  $= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Hamming Codes

- Class of *error-correcting* codes
- Capable of correcting all *single-error* patterns
- For each m ≥ 2, there is a Hamming code of length $n=2^m-1$ with $n-k=m$ parity check bits

Redundancy

m	$n=2^m-1$	k=n-m	m/n
3	7	4	3/7
4	15	11	4/15
5	31	26	5/31
6	63	57	6/63

m=3 Hamming Code

- Information bits are b_1, b_2, b_3, b_4
- Equations for parity checks b_5, b_6, b_7

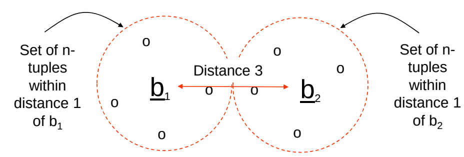
$$b_5 = b_1 \quad + b_3 + b_4$$
$$b_6 = b_1 + b_2 \quad + b_4$$
$$b_7 = \quad + b_2 + b_3 + b_4$$

- There are 2^4=16 codewords
- (0,0,0,0,0,0,0) is a codeword

Hamming (7,4) code

Information				Codeword							Weight
b_1	b_2	b_3	b_4	b_1	b_2	b_3	b_4	b_5	b_6	b_7	$w(\underline{b})$
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1	1	1	4
0	0	1	0	0	0	1	0	1	0	1	3
0	0	1	1	0	0	1	1	0	1	0	3
0	1	0	0	0	1	0	0	0	1	1	3
0	1	0	1	0	1	0	1	1	0	0	3
0	1	1	0	0	1	1	0	1	1	0	4
0	1	1	1	0	1	1	1	0	0	1	4
1	0	0	0	1	0	0	0	1	1	0	3
1	0	0	1	1	0	0	1	0	0	1	3
1	0	1	0	1	0	1	0	0	1	1	4
1	0	1	1	1	0	1	1	1	0	0	4
1	1	0	0	1	1	0	0	1	0	1	4
1	1	0	1	1	1	0	1	0	1	0	4
1	1	1	0	1	1	1	0	0	0	0	3
1	1	1	1	1	1	1	1	1	1	1	7

Minimum distance of any Hamming Code = 3



- Spheres of distance 1 around each codeword do not overlap
- If a single error occurs, the resulting n-tuple will be in a unique sphere around the original codeword
- Thus, receiver can correct erroneous reception back to original codeword

Outline Physical Layer Lectures

- Digital representation of information
  - Digital representation of analog signals
- Analog versus digital transmission
- Basic properties of dig. transmission
- Fundamental limits of dig. transmission
- Line coding, modulation
  - Amplitude, frequency, and phase modulation
- Properties of transmission media
- Synchronization
- Error detection and error correction