CS/ECE 439: Wireless Networking

Physical Layer - Diversity

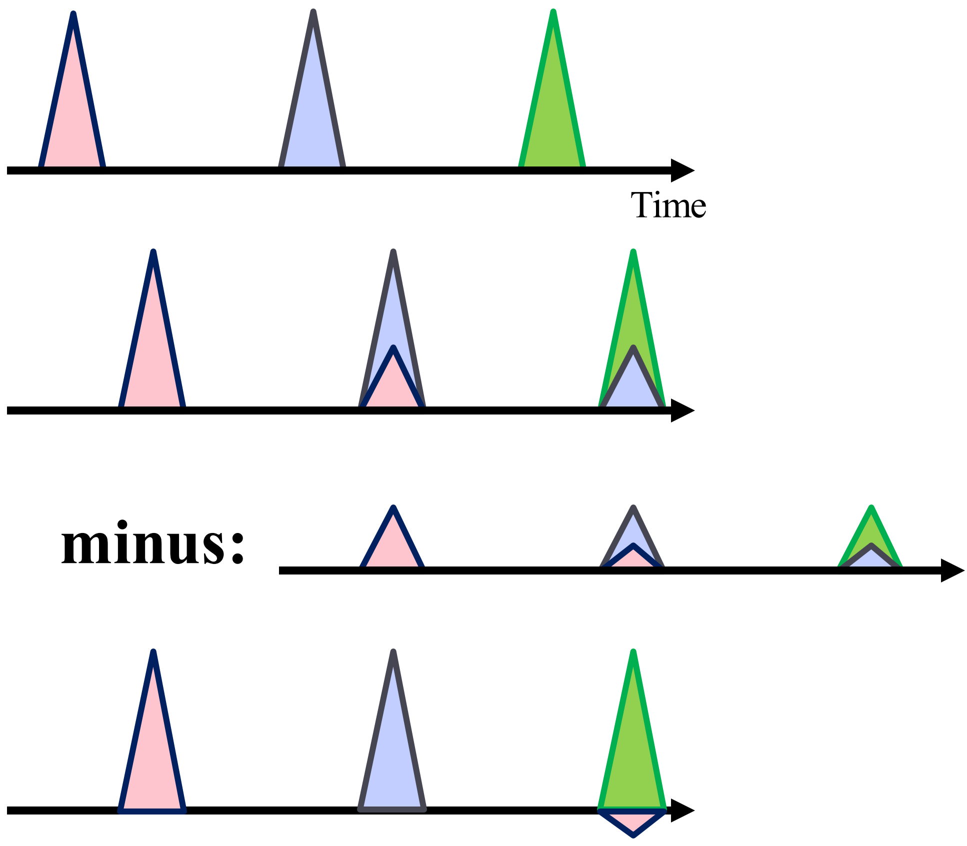
 Fall 2025

GRAINGER ENGINEERING

COMPUTER SCIENCE

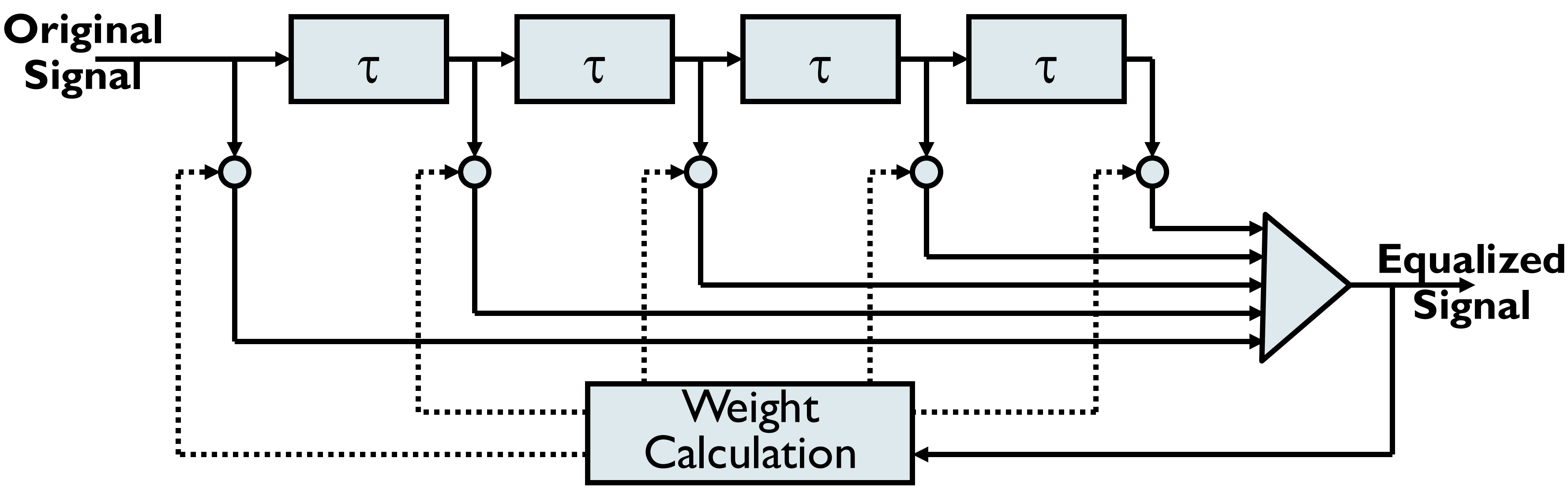
Inter-Symbol Interference

- Larger difference in path length can cause inter-symbol interference (ISI)
- Suppose the receiver can do some processing
  - Add/subtracted scaled and delayed copies of the signal

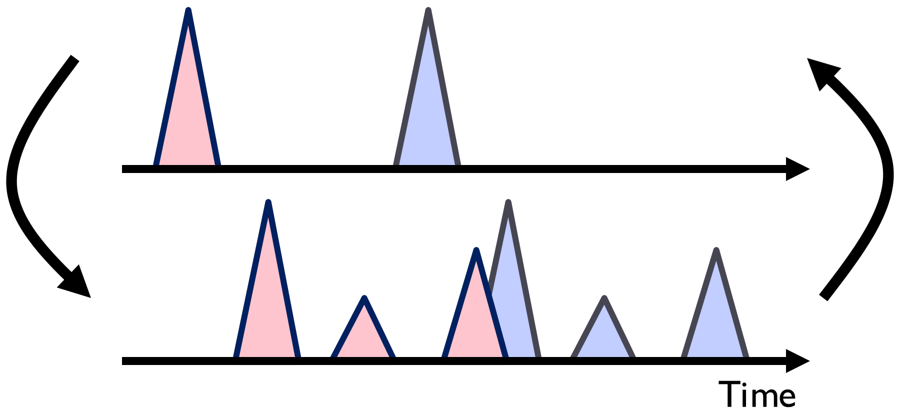


Dynamic Equalization

- Combine multiple delayed copies of the signal
  - ex: linear equalizer circuit



Equalization Discussion



- Use multiple delayed copies of the received signal to try to reconstruct the original signal
- Weights are set dynamically
  - Typically based on some known "training" sequence
- Effectively uses the multiple copies of the signal to reinforce each other
  - But only works for paths that differ in length by less than the depth of the pipeline

Diversity Techniques

- Spatial diversity
  - Exploit fact that fading is location-specific
  - Use multiple nearby antennas and combine signals
    - Can be directional
- Frequency diversity
  - Spread signal over multiple frequencies/broader frequency band
    - For example, spread spectrum
- Channel Diversity
  - Distribute signal over multiple "channels"
    - "Channels" experience independent fading
    - Reduces the error, i.e. only part of the signal is affected
- Time diversity
  - Spread data out over time
  - Expand bit stream into a richer digital signal
    - Useful for bursty errors, e.g. slow fading
    - A specific form of channel coding

Spatial Diversity

- Use multiple antennas that pick up the signal in slightly different locations
  - Can use more than two antennas!
- Each antenna experiences different channels
  - If antennas are sufficiently separated, chances are that the signals are mostly uncorrelated
  - If one antenna experiences deep fading, chances are that the other antenna has a strong signal
    - Antennas should be separated by ½ wavelength or more
- Applies to both transmit and receive side
  - Channels are symmetric

Receiver Diversity

- Simplest solution
  - Selection diversity: pick antenna with best SNR

- But why not use both signals?

  + More information

  - Signals out of phase, e.g. kind of like multi-path

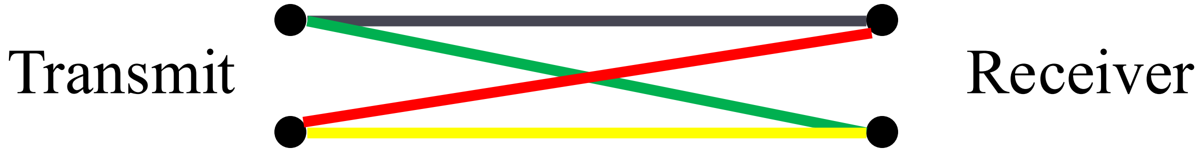
  ? Don't amplify the noise
- Maximal ratio combining: combine signals with a weight that is based on their SNR
  - Weight will favor the strongest signal (highest SNR)

Transmit Diversity

- Same as receive diversity but the transmitter has multiple antennas
- Selection diversity: transmitter picks the best antenna
  - i.e. with best channel to receiver
  - Sender "precodes" the signal
- How does transmitter learn channel?
  - Gets explicit feedback from the receiver
  - Rely on channel reciprocity

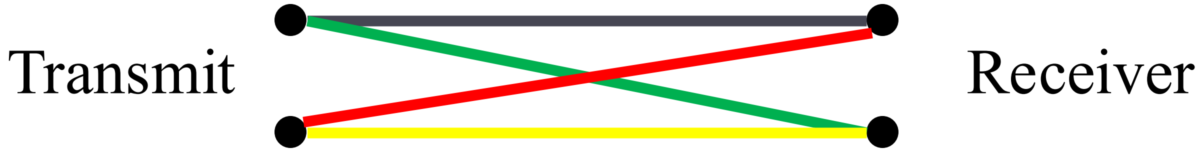
Typical Algorithm in 802.11

- Use transmit + receive selection diversity
- How to explore all channels to find the best one … or at least the best transmit antenna
- Receiver
  - Use the antenna with the strongest signal
  - Always use the same antenna to send the acknowledgement – gives feedback to the sender



Typical Algorithm in 802.11

- Use transmit + receive selection diversity
- How to explore all channels to find the best one … or at least the best transmit antenna
- Sender
  - Pick an antenna to transmit and learn about the channel quality based on the ACK
  - Occasionally try the other antenna to explore the channel between all four channel pairs

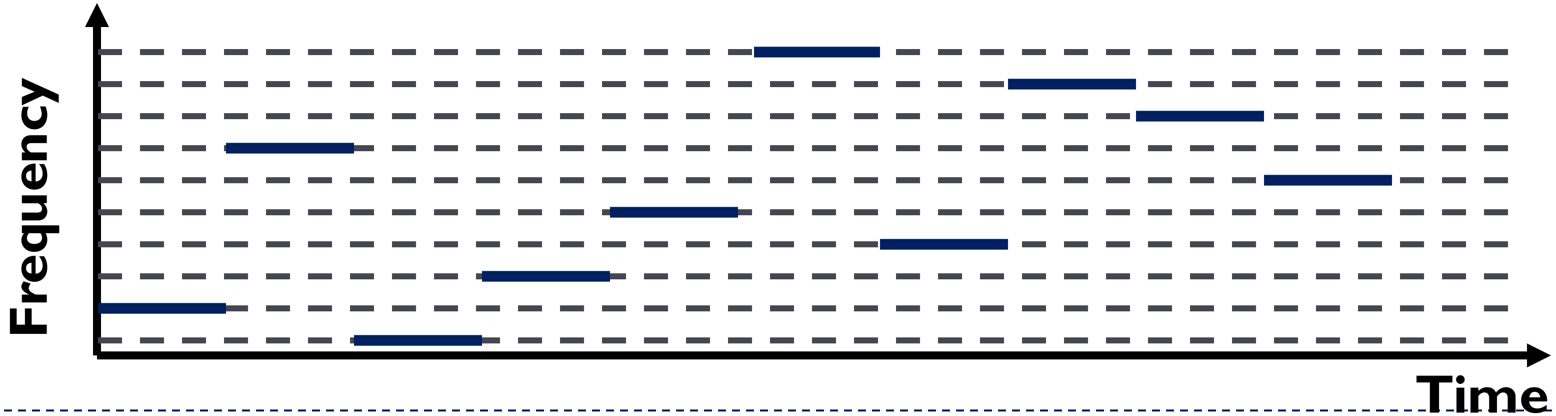


Spread Spectrum

- Spread transmission over a wider bandwidth
  - Don't put all your eggs in one basket!
  - Good for military
    - Jamming and interception becomes harder
  - Also useful to minimize impact of a "bad" frequency in regular environments
- What can be gained from this apparent waste of spectrum?
  - Immunity from various kinds of noise and multipath distortion
  - Can be used for hiding and encrypting signals
  - Several users can independently use the same higher bandwidth with very little interference

Frequency Hopping Spread Spectrum (FHSS)

- Have the transmitter hop between a seemingly random sequence of frequencies
  - Each frequency has the bandwidth of the original signal
- Dwell time is the time spent using one frequency
- Spreading code determines the hopping sequence
  - Must be shared by sender and receiver (e.g. standardized)



Example: Original 802.11 Standard (FH)

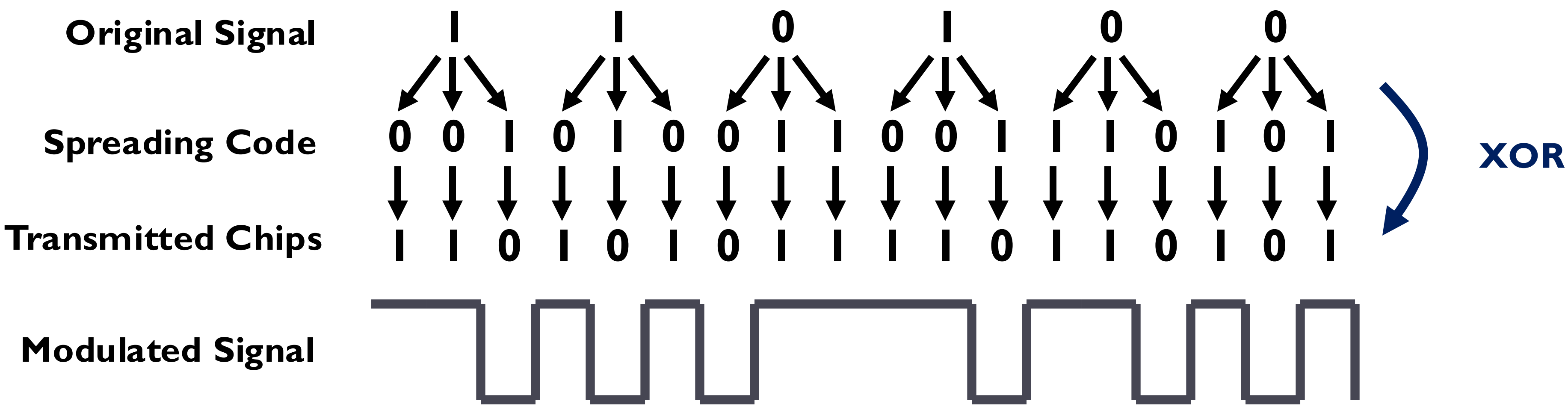
- 96 channels of 1 MHz
  - Only 78 used in US
    - Other countries used different numbers
  - Each channel carried only ~1% of the bandwidth
  - 1 or 2 Mbps per channel
- Dwell time was configurable
  - FCC set an upper bound of 400 msec
  - Transmitter/receiver must be synchronized
- Standard defined 26 orthogonal hop sequences
  - Transmitter used a beacon on fixed frequency to inform the receiver of its hop sequence
- Can support multiple simultaneous transmissions – use different hop sequences
  - e.g. up to 10 co-located APs with their clients

Example: Bluetooth

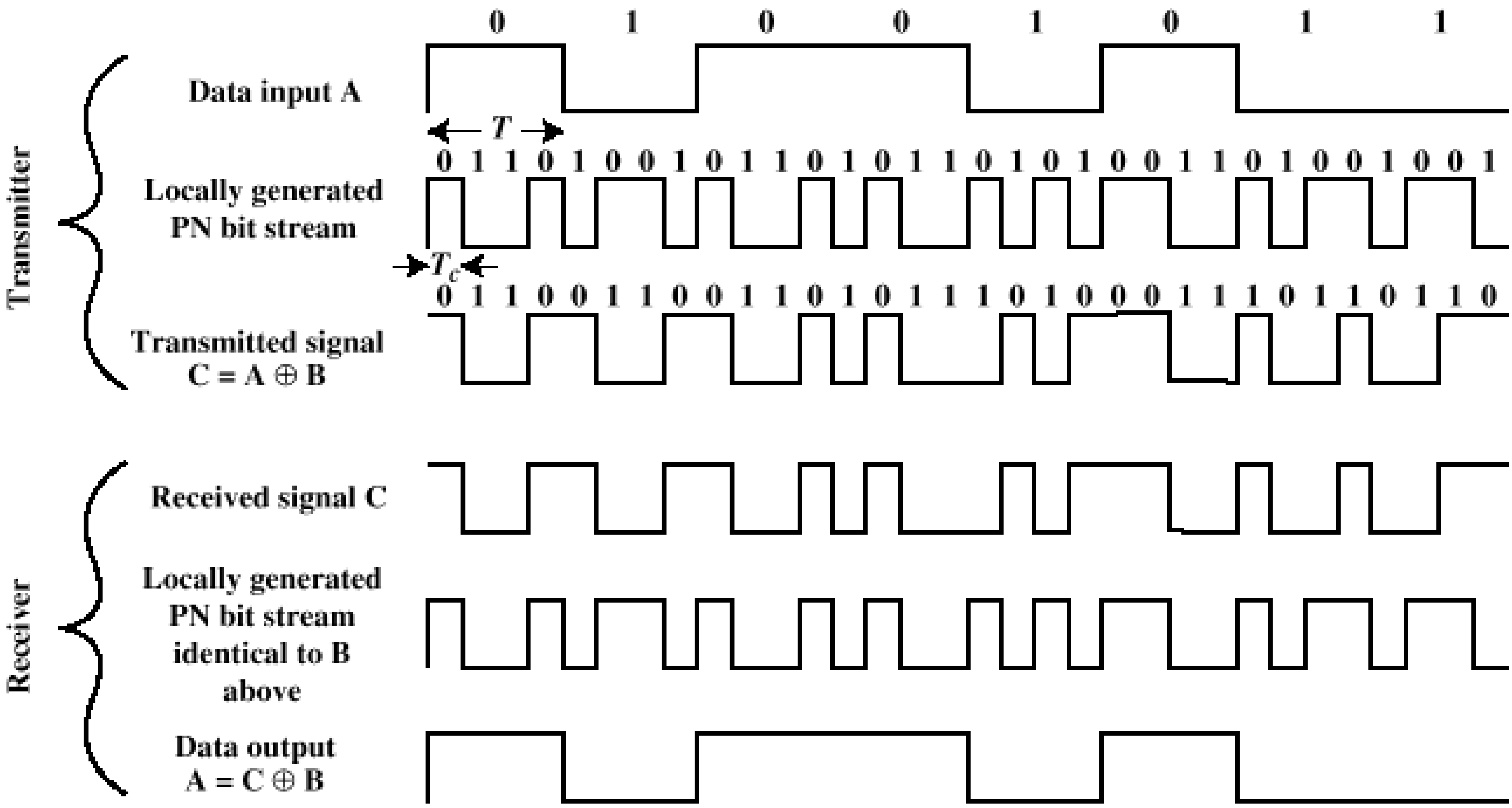
- 79 frequencies with a spacing of 1 MHz
  - Other countries use different numbers of frequencies
- Frequency hopping rate is 1600 hops/s
- Maximum data rate is 1 MHz

Direct Sequence Spread Spectrum (DSSS)

- Each bit in original signal is represented by multiple bits (chips) in the transmitted signal
- Spreading code spreads signal across a wider frequency band
  - Spread is in direct proportion to number of bits used
  - e.g. exclusive-OR of the bits with the spreading code
- The resulting bit stream is used to modulate the signal



Direct Sequence Spread Spectrum (DSSS)



Properties

- Each bit is sent as multiple chips
  - Need more bps bandwidth to send signal
  - Number of chips per bit = spreading ratio
    - This is the spreading part of spread spectrum
- Need more spectral bandwidth
  - Nyquist and Shannon say so!
- Advantages
  - Transmission is more resilient.
    - DSSS signal will look like noise in a narrow band
    - Can lose some chips in a word and recover easily
  - Multiple users can share bandwidth

Example: Original 802.11 Standard (DSSS)

- DSSS PHY
  - 1 Msymbol/s rate
  - 11-to-1 spreading ratio
  - Barker chipping sequence
    - Barker sequence has low autocorrelation properties
      - The similarity between observations as a function of the time lag between them
  - Uses about 22 MHz
- Receiver decodes by counting the number of "1" bits in each word
  - 6 "1" bits correspond to a 0 data bit
- Data rate
  - 1 Mbps (i.e. 11 Mchips/sec)
  - Extended to 2 Mbps
    - Requires the detection of a ¼ phase shift

Example: 802.11b

- (Maximum) data rate
  - 11 Mbs
- Complementary Code Keying (CCK)
  - Complementary means that the code has good auto-correlation properties
    - Want nice properties to ease recovery in the presence of noise, multipath interference, ..
  - Each word is mapped onto an 8 bit chip sequence
  - Symbol rate at 1.375 MSymbols/sec, at 8 bpS = 11 Mbps
- Symbol rate
  - 1.375 MSymbols/sec, at 8 bpS = 11 Mbps

Code Division Multiple Access

- Users share spectrum and time, but use different codes to spread their data over frequencies
  - DSSS where users use different spreading sequences
  - Use spreading sequences that are orthogonal, i.e. they have minimal overlap
  - Frequency hopping with different hop sequences
- The idea is that users will only rarely overlap and the inherent robustness of DSSS will allow users to recover if there is a conflict
  - Overlap = use the same the frequency at the same time
  - The signal of other users will appear as noise

CDMA Principle

- Basic Principles of CDMA
  - D = rate of data signal
  - Break each bit into k chips - user-specific fixed pattern
  - Chip data rate of new channel = kD
- If k=6 and code is a sequence of 1s and -1s
  - For a '1' bit, A sends code as chip pattern
    - <c1, c2, c3, c4, c5, c6>
  - For a '0' bit, A sends complement of code
    - <-c1, -c2, -c3, -c4, -c5, -c6>
- Receiver knows sender's code and performs electronic decode function

$$S_u(d) = d1 \times c1 + d2 \times c2 + d3 \times c3 + d4 \times c4 + d5 \times c5 + d6 \times c6$$

  - <d1, d2, d3, d4, d5, d6> = received chip pattern
  - <c1, c2, c3, c4, c5, c6> = sender's code

CDMA Example

- User A code = <1, –1, –1, 1, –1, 1>
  - To send a 1 bit = <1, –1, –1, 1, –1, 1>
  - To send a 0 bit = <–1, 1, 1, –1, 1, –1>
- User B code = <1, 1, –1, – 1, 1, 1>
  - To send a 1 bit = <1, 1, –1, –1, 1, 1>
- Receiver receiving with A's code
  - (A's code) x (received chip pattern)
    - User A '1' bit: 6 -> 1
    - User A '0' bit: -6 -> 0
    - User B '1' bit: 0 -> unwanted signal ignored

CDMA Example

- CDMA cellular standard
  - Used in the US, e.g. Sprint
- Allocates 1.228 MHz for base station to mobile communication
  - Shared by 64 "code channels"
  - Used for voice (55), paging service (8), and control (1)
- Provides a lot error coding to recover from errors
  - Voice data is 8550 bps
  - Coding and FEC increase this to 19.2 kbps
  - Then spread out over 1.228 MHz using DSSS; uses QPSK

Discussion

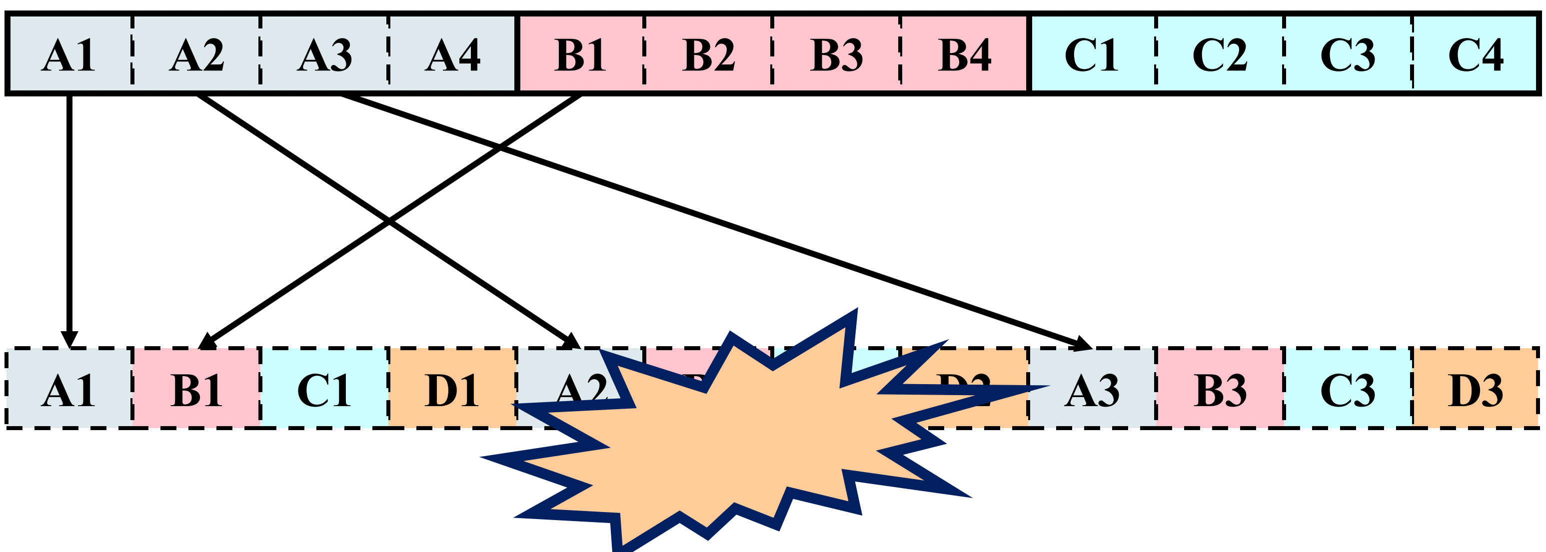
- Spread spectrum is very widely used
- Effective against noise and multipath
  - Signal looks like noise to other nodes
  - Multiple transmitters can use the same frequency range
- FCC requires the use of spread spectrum in ISM band
  - If signal is above a certain power level
- Is also used in higher speed 802.11 versions.
  - No surprise!

Time Redundancy: Bit Stream Level

- Protect digital data by introducing redundancy in the transmitted data
  - Error detection codes: can identify certain types of errors
  - Error correction codes: can fix certain types of errors
- Block codes provide Forward Error Correction (FEC) for blocks of data
  - (n, k) code: n bits are transmitted for k information bits
  - Simplest example: parity codes
  - Many different codes exist: Hamming, cyclic, Reed-Solomon, …
- Convolutional codes provide protection for a continuous stream of bits
  - Coding gain is n/k
  - Turbo codes: convolutional code with channel estimation

Time Diversity Example

- Spread blocks of bytes out over time
- Can use FEC or other error recovery techniques to deal with burst errors



Error Detection/Recovery

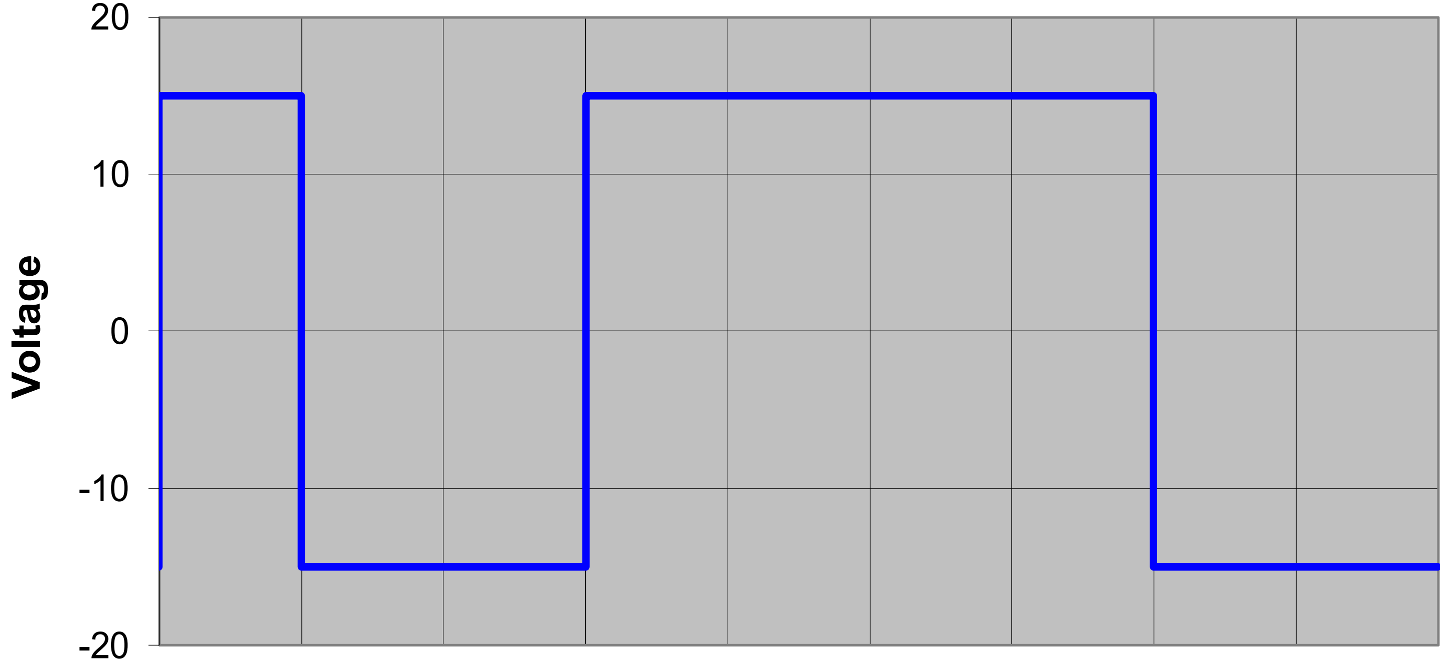
- Adds redundant information that checks for errors
  - And potentially fix them
  - If not, discard packet and resend
- Occurs at many levels
  - Demodulation of signals into symbols (analog)
  - Bit error detection/correction (digital)—our main focus
    - Within network adapter (CRC check)

Error Detection/Recovery

- Analog Errors
  - Example of signal distortion
- Hamming distance
  - Parity and voting
  - Hamming codes
- Error bits or error bursts?
- Digital error detection
  - Two-dimensional parity
  - Cyclic Redundancy Check (CRC)

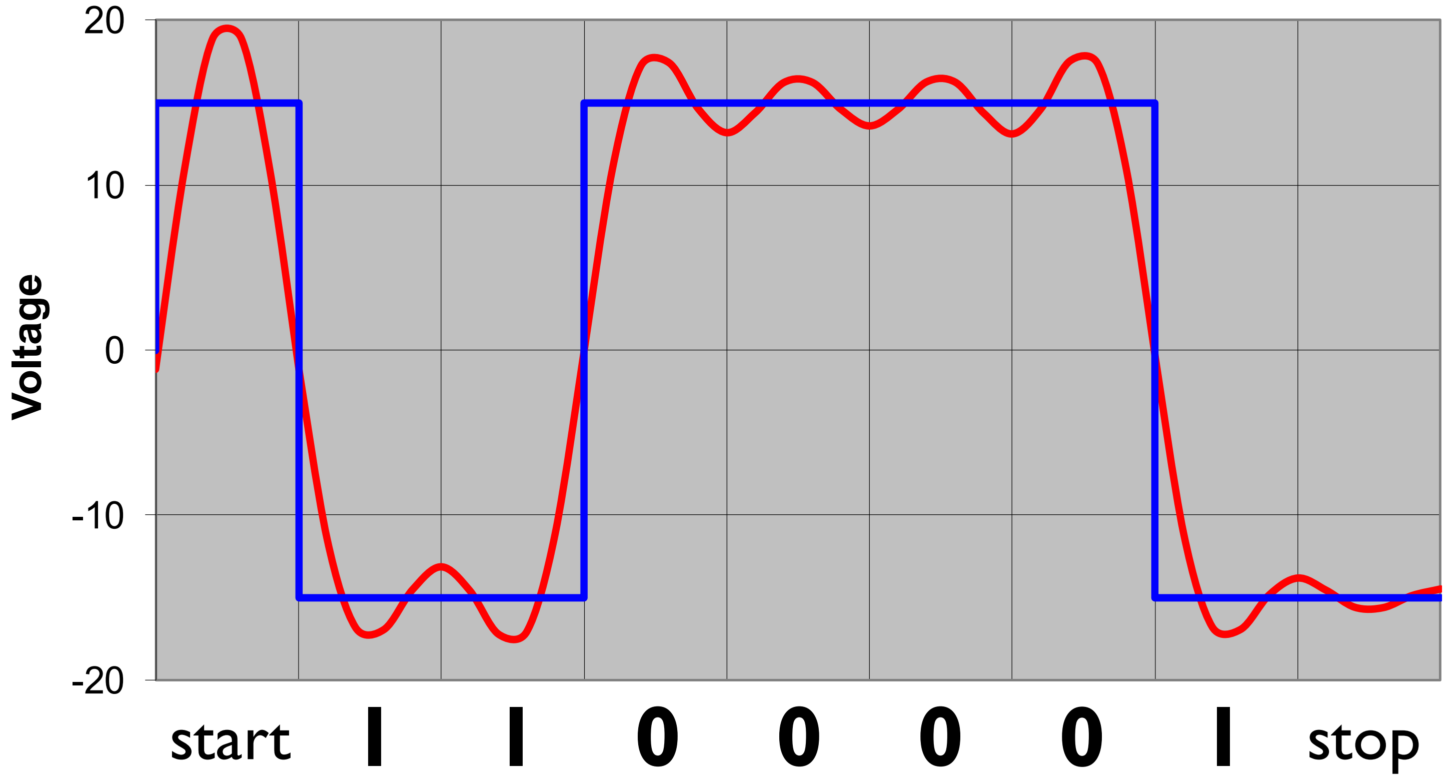
Analog Errors

- Consider the following encoding of ‘Q’

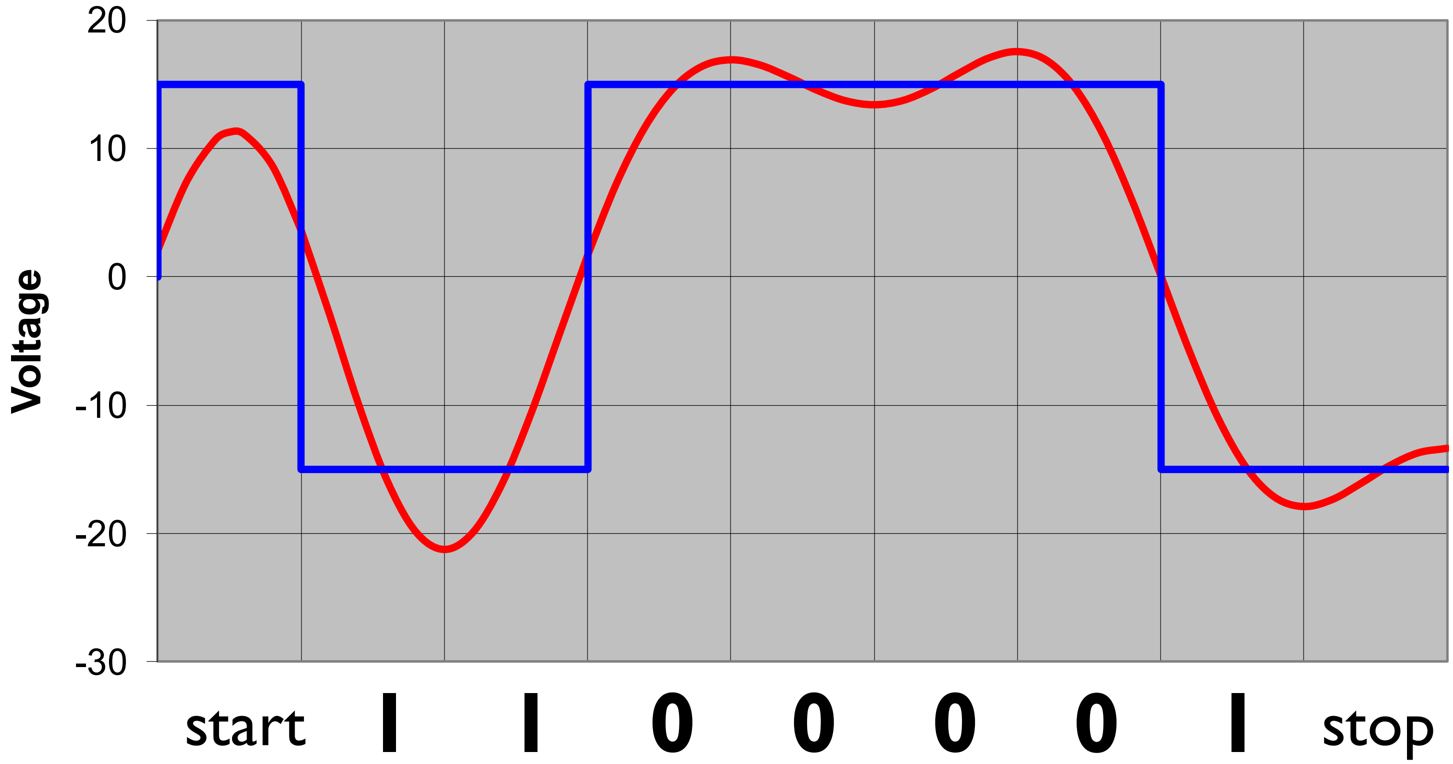


start 1 1 0 0 0 0 1 stop

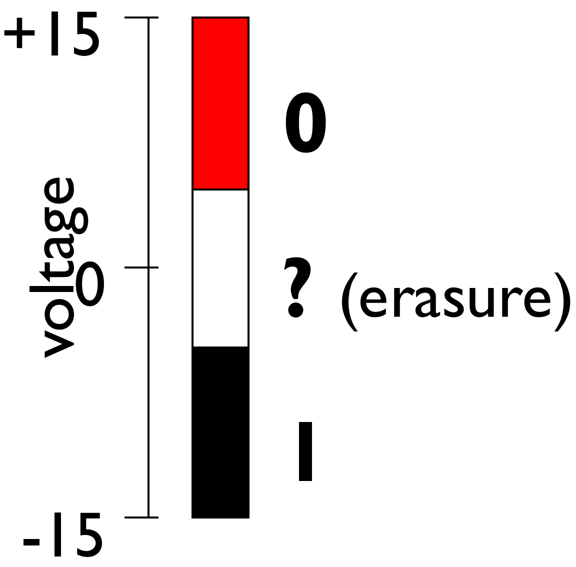
Encoding isn't perfect



Encoding isn't perfect

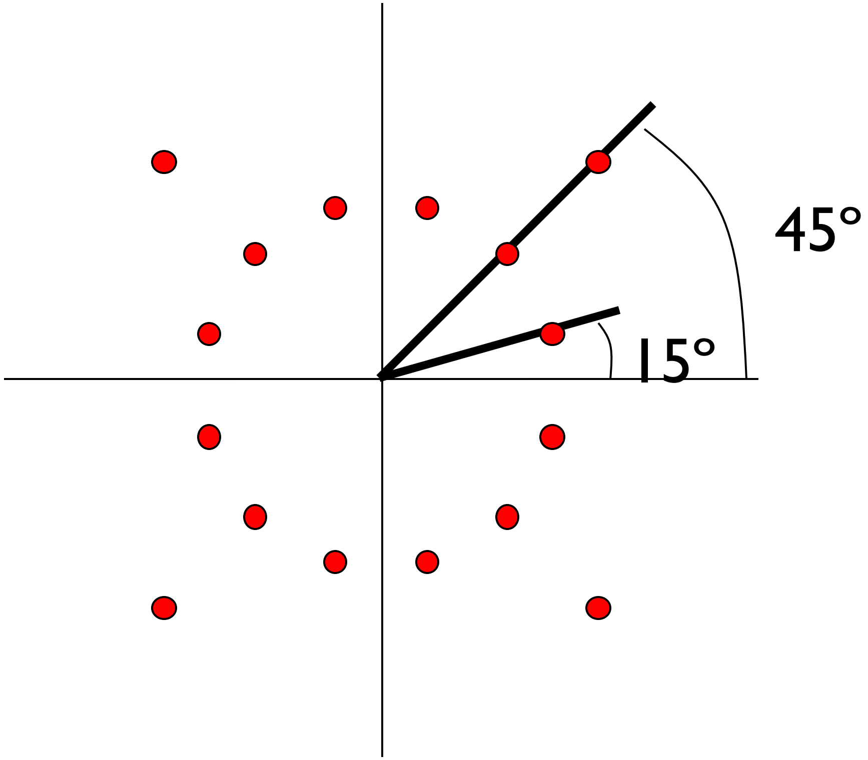


Symbols



possible binary voltage encoding symbol neighborhoods and erasure region

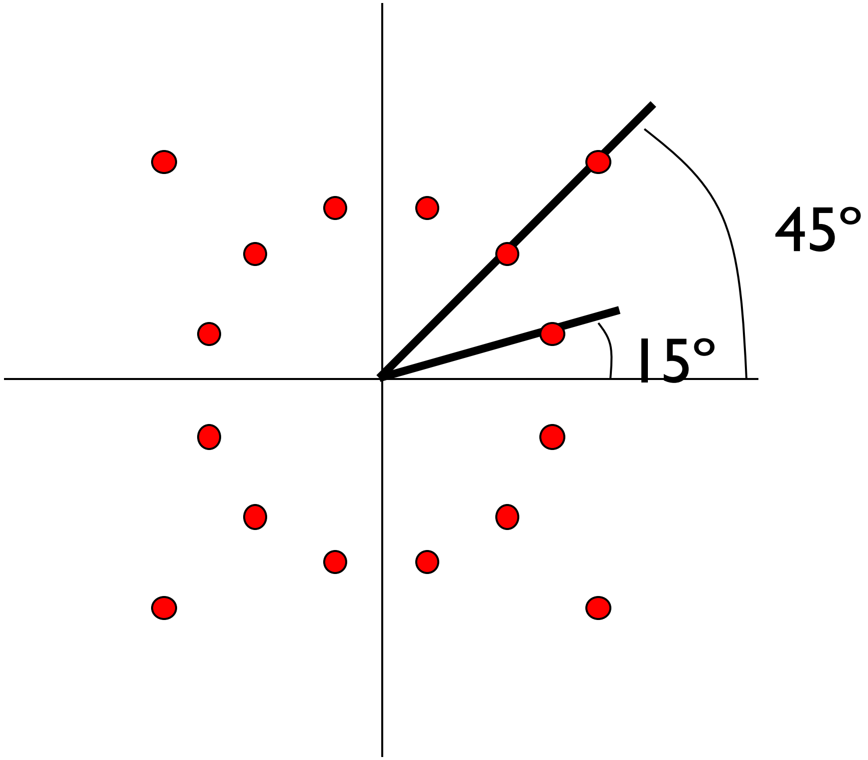
Symbols



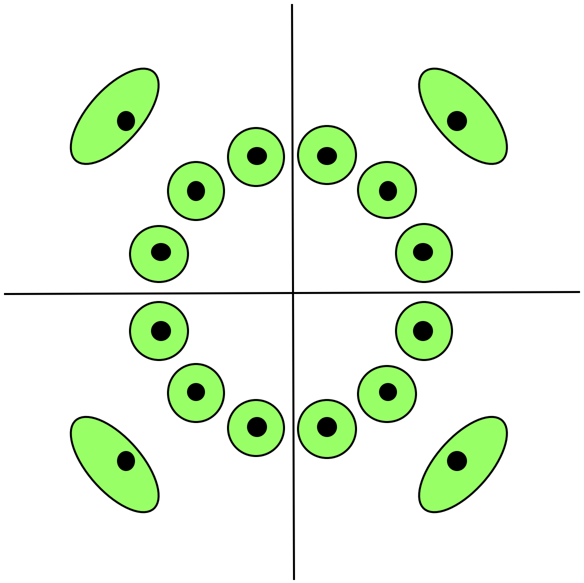
16-symbol example

- QAM
  - Phase and amplitude modulation
- 2-dimensional representation
  - Angle is phase shift
  - Radial distance is new amplitude

Symbols



16-symbol example



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

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

$$P_2 = 1 - P_1$$

$$P_3 = 0$$

  - 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

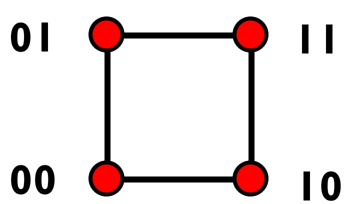
Parity

- 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

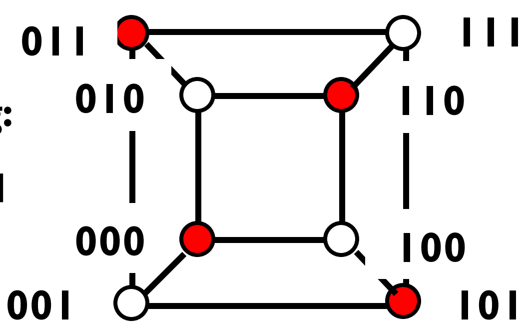
- 1-bit error detection with parity
  - Add an extra bit to a code to ensure an even (odd) number of 1s
  - Every code word has an even (odd) number of 1s

Valid code words



Parity Encoding:

White – invalid (error)



Voting: Correcting Bit Flips

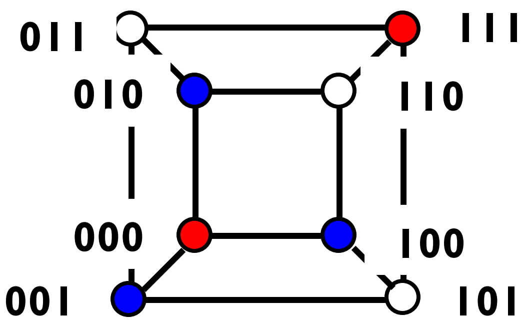
- 1-bit error correction with voting
  - Every codeword is transmitted n times
  - Codeword is 3 bits long

Valid code words

Voting:

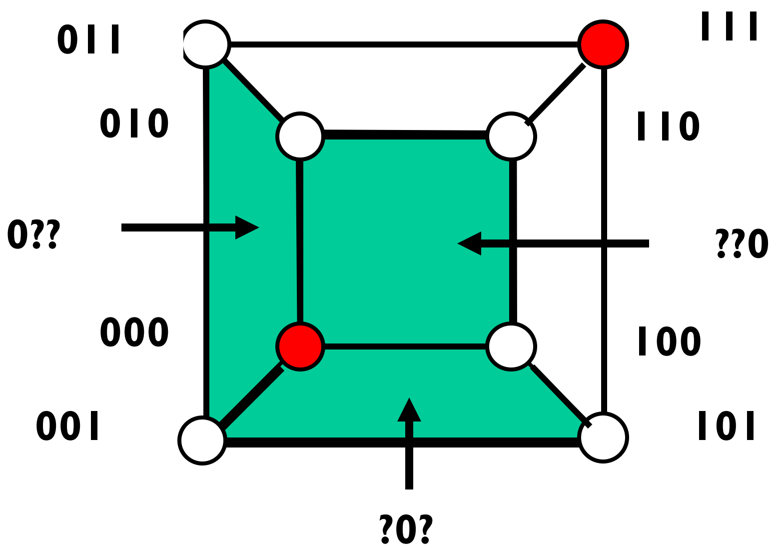
White – correct to 1

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

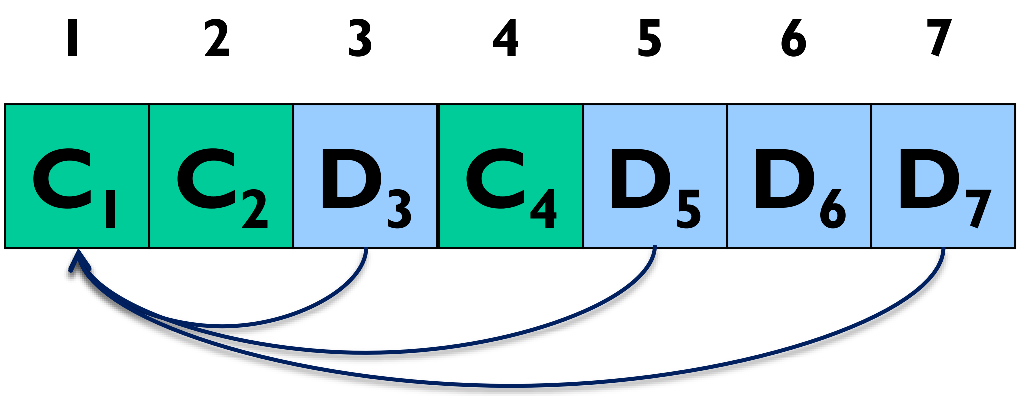
- N-bit error detection
  - No code word changed into another code word
  - Requires Hamming distance of N+1
- N-bit error correction
  - N-bit neighborhood: all codewords within N bit flips
  - No overlap between N-bit neighborhoods
  - Requires hamming distance of 2N+1

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

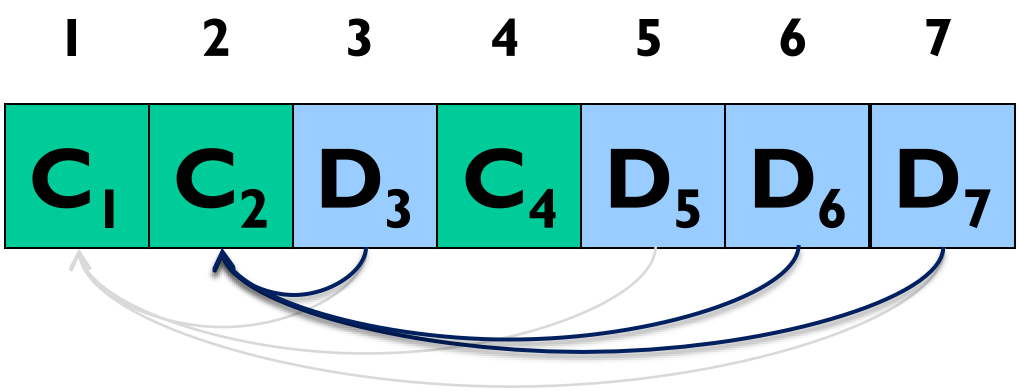
Hamming Codes

- 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



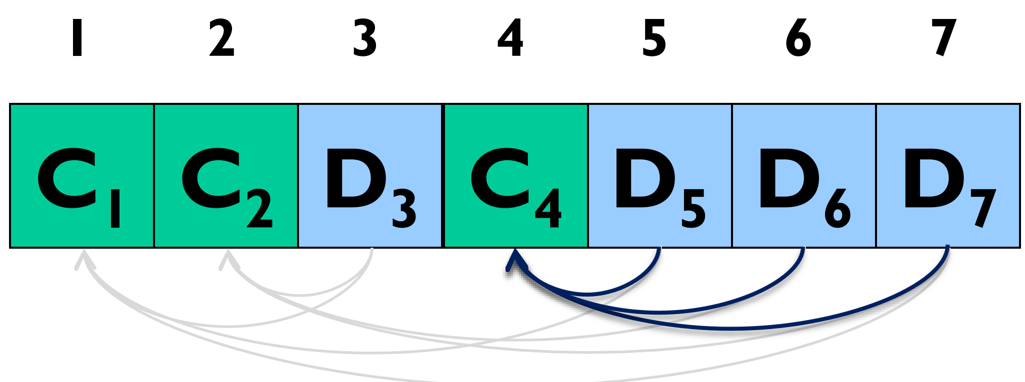
Hamming Codes

- 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

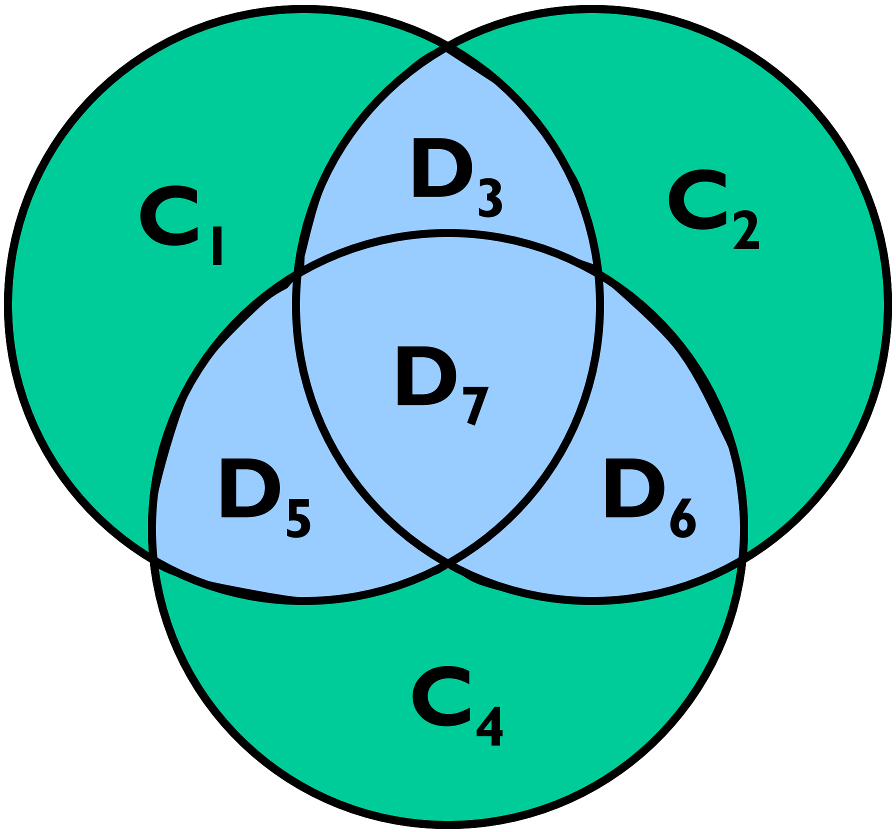


Hamming Codes

- 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



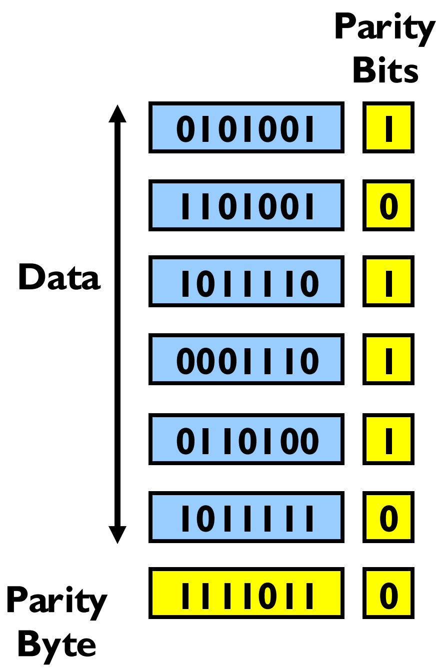
What are we trying to handle?

- 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



- 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 happens if…

Can detect exactly which bit flipped
Can also correct it!

0	1	~~0~~ 1	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 2-bit errors?

Can detect the two-bit error

Can't detect a problem here

Can't tell which bits are flipped, so can't correct

0	1	~~0~~ 1	0	0	~~1~~ 0	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 2-bit errors?

Could be the dotted pair or the dashed pair.
Can’t correct 2-bit error.

If these four parity bits don’t match
Which bits could be in error?

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 3-bit errors?

Can detect the three-bit error

0	1	~~0~~ 1	0	0	~~1~~ 0	1	~~1~~ 0	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

But you can't correct (eg if dashed bits got flipped instead of the dotted ones)

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

What about 4-bit errors?

Can you think of a 4-bit error this scheme can't detect?

~~0~~ 1	1	~~0~~ 1	0	0	1	1	1	0
0	1	1	0	1	1	1	1	0
~~0~~ 1	1	~~1~~ 0	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?

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 $M(x)$

    $= 1 * x^7 + 0 * x^6 + 0 * x^5 + 1 * x^4 + 1 * x^3 + 0 * x^2 + 1 * x + 0$

    $= x^7 + x^4 + x^3 + x$
- 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 $P(x) = M(x) \bullet x^k + R(x)$
- So $R(x) =$ remainder of $M(x) \bullet x^k / C(x)$
  - 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 (the divisor 1101)

Remainder (101)

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    <-- Received message
             1101
             ----
              1000
              1101
              ----
               1011
               1101
               ----
                1101
                1101
                ----
                 0101
```

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

Two bit errors

Received message

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
    -------
000111111110
```

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