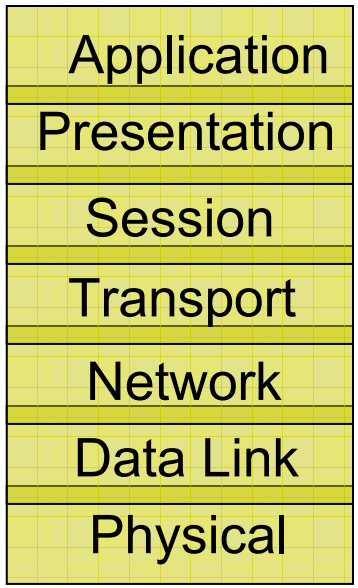
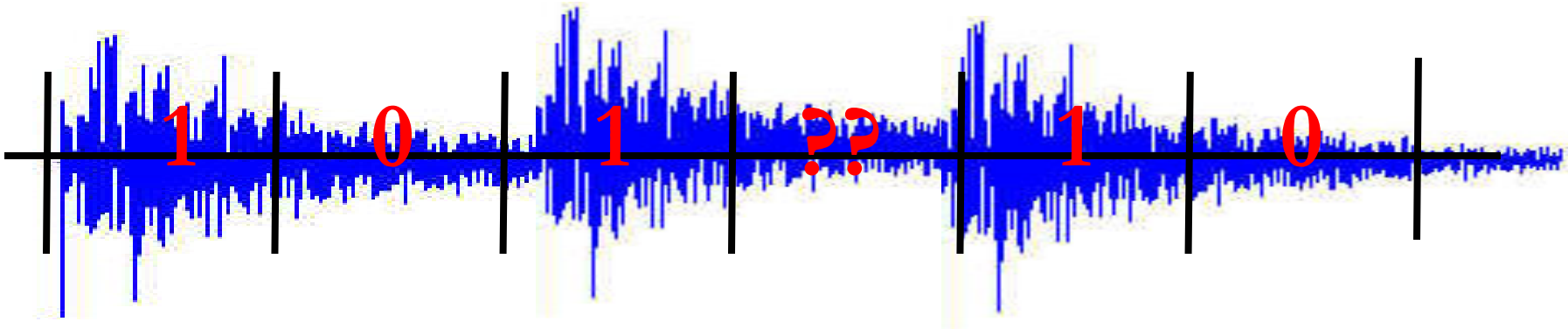
Codes for Error Detection/Correction

- Error detection and correction
  - How do we detect and correct messages that are garbled during transmission?
- The responsibility for doing this cuts across the different layers
  - But we're mostly thinking about links right now



Problem and Approach

- Noise can flip some of the bits we receive
  - We must be able to detect when this occurs!



- Basic approach: add redundant data
  - Error detection codes allow errors to be recognized
  - Error correction codes allow errors to be repaired too

Motivating Example

- A simple error detection scheme:
  - Just send two copies. Differences imply errors.

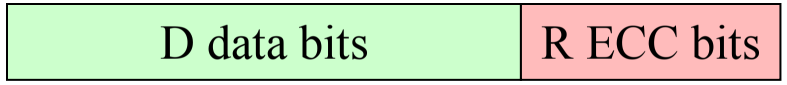
- Question: Can we do any better?
  - With less overhead
  - Catch more kinds of errors
- Answer: Yes – stronger protection with fewer bits
  - But we can't catch all inadvertent errors, nor malicious ones

- We will look at basic block codes
  - K bits in, N bits out is a (N,K) code
  - Simple, memoryless mapping

- Other important scheme in practice:
  - Convolutional codes, stream of bits in/out

Error Detection/Correction Codes

- Detection/correction schemes are characterized in two ways:
  - Overhead: ratio of total bits sent to data bits, minus 1
    - Example: 1000 data bits + 100 code bits = 10% overhead
  - The errors they detect/correct
    - E.g., all single-bit errors, all bursts of fewer than 3 bits, etc.
- A scheme maps D bits of data into D+R bits – i.e., it uses only 2^D distinct bit strings of the 2^{D+R} possible.



(systematic)

- The sender computes the ECC bits based on the data.
- The receiver also computes ECC bits for the data it receives and compares them with the ECC bits it received. Mismatches detect errors. Mapping to the closest valid codeword can correct errors.

Patterns of Errors Matter

- Q: Suppose you expect a bit error rate of about 1 bit per 1000 sent. What fraction of packets would be corrupted if they were 1000 bits long (and you could detect all errors but correct none)?

- A: It depends on the pattern of errors
  - Bit errors occur at random
    - Packet error rate is about $1 - 0.999^{1000} = 63\%$
  - Errors occur in bursts, e.g., 100 consecutive bits every 100,000 bits
    - Packet error rate ≤ 2%

Real Error Models

- Random, e.g., thermal noise as in AWGN
- Bursty, e.g., wires, if there is an error it is likely to be a burst
  - Common due to physical effects
- Errors can also be "erasures", but we will ignore

- For bursty errors, either want:
  - A code that is built to handle them well
  - To convert them to random errors (interleaving)

- Interleaving
  - Error-free code words: aaaabbbbccccddddeeeeffffgggg
    Interleaved: abcdefgabcdefgabcdefgabcdefg
  - Transmission w/ burst error: abcdefgabcd____bcdefgabcdefg
    Received w/ deinterleaving: aa_abbbbccccdddde_eef_ffg_gg

The Hamming Distance

- Errors must not turn one valid codeword into another valid codeword, or we cannot detect/correct them.
- Hamming distance of a code is the smallest number of bit differences that turn any one codeword into another
  - e.g, code 000 for 0, 111 for 1, Hamming distance is 3
- For code with distance d+1:
  - d errors can be detected, e.g, 001, 010, 110, 101, 011
- For code with distance 2d+1:
  - d errors can be corrected, e.g., 001 → 000

Parity

- Start with n bits and add another so that the total number of 1s is even (even parity)
  - e.g. 0110010 → 01100101
  - Easy to compute as XOR of all input bits
- Will detect an odd number of bit errors
  - But not an even number
- Does not correct any errors

Checksums

- Used in Internet protocols (IP, ICMP, TCP, UDP)
- Basic Idea: Add up the data and send it along with sum

- Algorithm:
  - Mouthful for "sum": "checksum is the 1s complement of the 1s complement sum of the data interpreted 16 bits at a time" (for 16-bit TCP/UDP checksum)
  - 1s complement nit: flip all bits to make a number negative, so adding requires carryout to be added back.

- Q: What kind of errors will checksums detect?

CRCs (Cyclic Redundancy Check)

- Stronger protection than checksums
  - Used widely in practice, e.g., Ethernet CRC-32
  - Implemented in hardware (XORs and shifts)

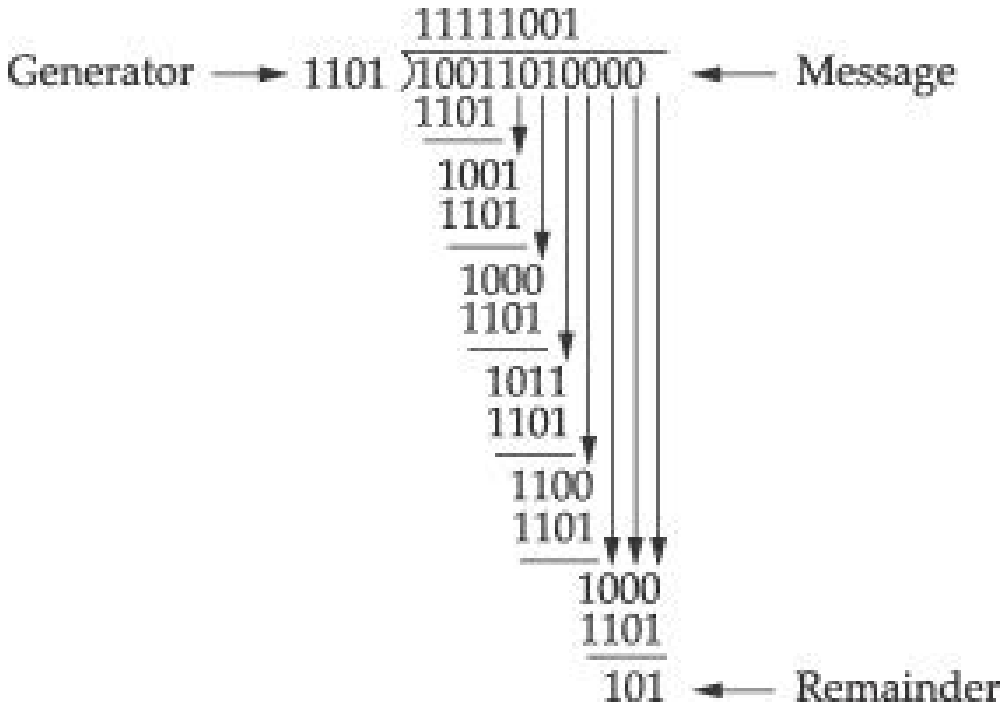
- Algorithm: Given n bits of data, generate a k bit check sequence that gives a combined n + k bits that are divisible by a chosen divisor C(x)

- Based on mathematics of finite fields
  - "numbers" correspond to polynomials, use modulo arithmetic
  - e.g, interpret 10011010 as $x^7 + x^4 + x^3 + x^1$

CRC Example

- Extend message with *k* 0's, when using a *k*-degree generator
- Divide message by generator (XOR)
- Discard result
- Subtract remainder from original message

- On reception, check that message is divisible by generator



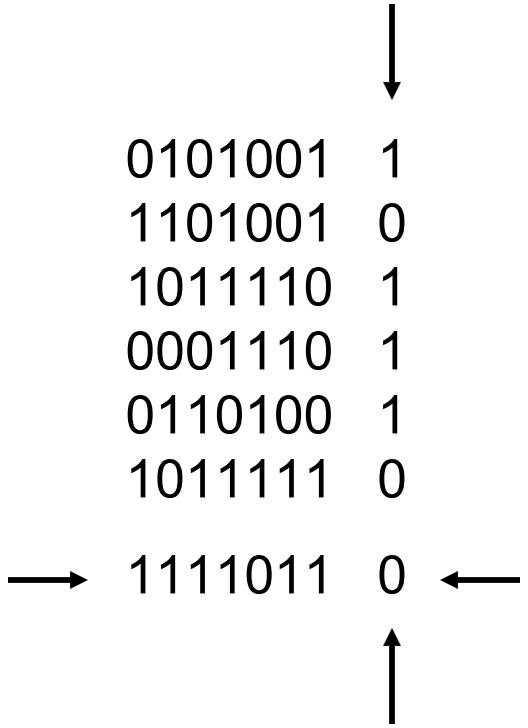
How is C(x) Chosen?

- Mathematical properties:
  - All 1-bit errors if non-zero x^k and x^0 terms
  - All 2-bit errors if C(x) has a factor with at least three terms
  - Any odd number of errors if C(x) has (x + 1) as a factor
  - Any burst error < k bits

- There are standardized polynomials of different degree that are known to catch many errors
  - Ethernet CRC-32: 100000100110000010001110110110111

2D Parity (just for example)

- Add parity row/column to array of bits
- How many simultaneous bit errors can it detect?
- Which errors can it correct?



Hamming Codes

- Learn about this in section
- Clever scheme of (roughly) parity bits across different subsets of the data bits that locate an error.
- Can correct single bit errors, detect double bit errors
- Used in some ECC memory

Reed-Solomon / BCH Codes

- Note: ECC often called FEC (Forward Error Correction) when used at the higher layers

- Developed to protect data on magnetic disks
- Used for CDs and cable modems too
- Work on symbols, e.g., 8 bits, not bits.
- Property: 2t redundant symbols can correct <= t errors
- Mathematics somewhat more involved …

Real Error Detection/Correction codes

- Detection
  - Checksums, but weak
  - CRCs, widely used
- Correction
  - Convolutional codes
  - Reed-Solomon / BCH
  - Low-density Parity Check (LDPC) codes
- Based on mathematical properties …

Error Correction

- Two strategies to correct errors:
  - Detect and retransmit, or Automatic Repeat reQuest. (ARQ)
  - Error correcting codes, or Forward Error Correction (FEC)
- Question: Which should we choose?

ARQ vs. FEC

- Will depend on the kind of errors and cost of recovery
- Example: Message with 1000 bits, Prob(bit error) 0.001
  - Case 1: random errors
  - Case 2: bursts of 1000 errors

- FEC used at low-level to lower residual error rate
- ARQ often used at packet level to fix large errors, e.g., collision, loss, as well as protect against residual errors
- FEC sometimes used at high level, e.g.:
  - Real time applications (no time to retransmit!)
  - Nice interaction with broadcast (different receiver errors!)