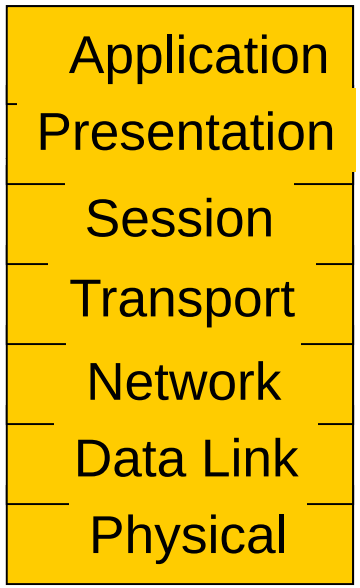
CSE 561 – Error detection & correction

David Wetherall
djw@cs.washington.edu

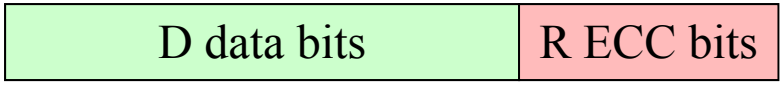
Codes for Error Detection/Correction

- 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



Error Detection/Correction Codes

- 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. And mapping to the closest valid codeword can correct errors.
- 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.

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

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/won't checksums detect?

Internet checksum properties

- Catches all error bursts up to 15 bits, most 16 bits
- Random errors detected with prob 1 – 2^-16

- Fails to catch transpositions, insertion/deletion of zeros, combinations
  - These are typically hardware/software bugs not random errors

Fletcher – a better checksum

- Includes a "positional component"

$$\begin{aligned} &\text{Initial values :} && sumA = sumB = 0; \\ &\text{For increasing } i : && \{\, sumA = sumA + D_i; \\ & && \quad sumB = sumB + sumA; \,\}. \end{aligned}$$

- Now sensitive to order of data
  - slightly more computation, but well worth it

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 $x^7 + x^4 + x^3 + x^1$
- Q: What kind of errors will/won't checksums detect?

“Standard” CRC-32

- It is
  - CRC-32: 100000100110000010001110110110111
  - Used for Ethernet, cable modems, ADSL, PPP, …
- Catches
  - All 1 and 2 bit errors
  - All burst errors < 32 bits
  - All errors with an odd number of flips
  - All based on mathematical properties; look in the book
  - Random errors with prob 1- 2^-32
- Stronger than checksums

A better CRC

- Castagnoli, Koopman
  - Via exhaustiv search!

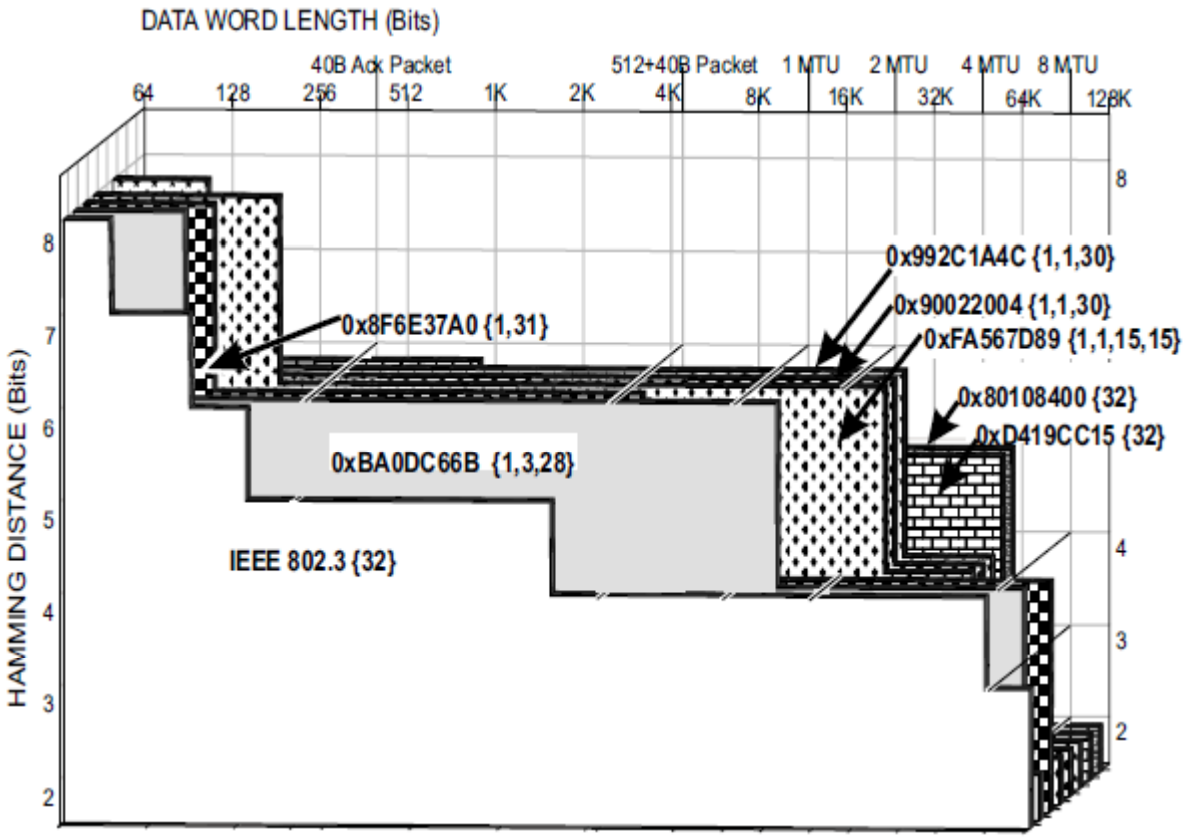


Figure 1. Error detection capabilities of selected 32-bit CRC polynomials.

Real Error Detection/Correction codes

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

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 $\leq 2\%$

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”, e.g., lost packet

- 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

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!)

802.11 error detection/correction

- The standard scheme is:

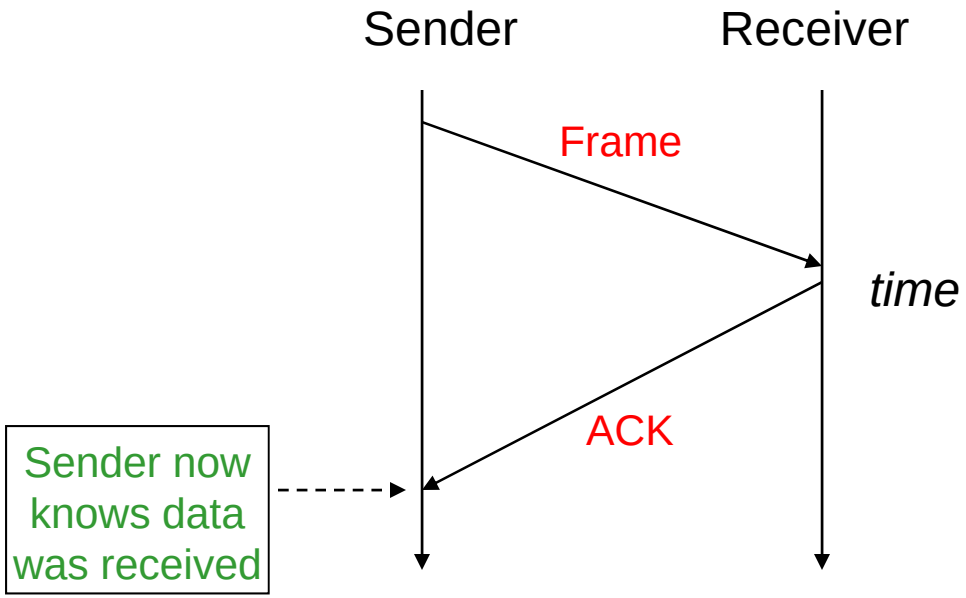
- Link: 32 bit CRC on frame and retransmission
- PHY header has 16 bit CRC
- PHY: FEC on data via interleaving and a binary convolutional code (or other options)
  - rates from ½ to 5/6.

Maranello

- What is the goal of the scheme?
- Why are there all these errors to correct anyway?
- How does the scheme work?
- What is the scheme assuming about errors?
- What are the performance costs?
- What are the performance benefits?
- How else might we design “partial packet recovery”?

Reliable Transmission

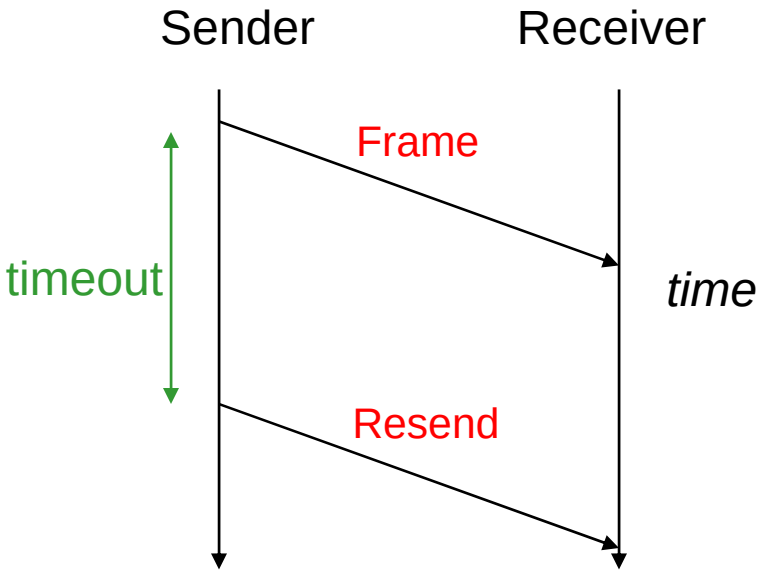
- Because there may be uncorrectable errors (no matter what ECC scheme is used), how can the sender be sure that the receiver got the data?
  - The sender must receive an acknowledgement (ACK) from the sender



What if noACK is received?

Timeouts / Automatic Repeat Request (ARQ)

- If no ACK comes back, the sender must re-send the data (ARQ)
  - When is the sender sure that no ACK is coming back?
    - As a practical matter delays are difficult to bound except for direct links
    - Sender chooses some reasonable timeout – if the ACK isn't back in that much time, it assumes it will never see an ACK, and re-sends



What if original frame arrived, but ACK was lost?

Duplicate Detection: Sequence Numbers

- So that the receiver can detect (and discard) duplicates, distinct frames are given distinct sequence numbers
  - E.g., 0, 1, 2, 3, …
- When a frame is re-sent, it is re-sent with the same sequence number as the original
- The receiver keeps some information about what sequence numbers it has seen, and discards arriving packets that are duplicates