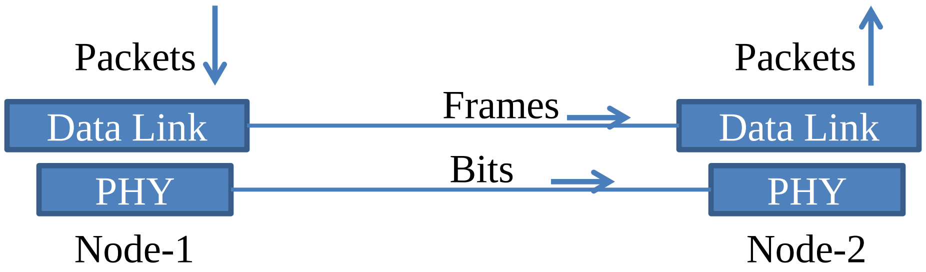
Data Link Layer: Error Detection

Kameswari Chebrolu

Recap

- Frame-by-Frame next-hop delivery
- Covered framing and overview of error-control
  - Hamming distance of a code determines the error detection and correction capabilities



General Approach

- Add redundant information to a frame
- At Sender:
  - Add **k** bits of redundant data to a **m** bit message
  - k << m; k = 32; m = 12,000 for Ethernet
  - k derived from original message through some algorithm
- At Receiver:
  - Reapply same algorithm as sender to detect errors; take corrective action if necessary

Parity Bit

- Even Parity: 1100, send 1100<u>0</u>
- Detects odd number of errors

Two Dimensional Parity

1101001	0
1011110	1
1001000	0
1111001	1
0000110	0

Parity Bits

Data

- Used by BISYNC protocol for ASCII characters
- "N + 8" bits of redundancy for "N" ASCII characters (character is 7 bits)

  it can correct 1 bit errors.
- Catches all 1, 2, 3 bit errors and most 4 bit errors

Internet Checksum

1's complement addition
  1 0 1 0
  1 1 0 0
=======
1 0 1 1 0
add carry back again
  0 1 1 1

- Used at the network layer (IP header)
- Algorithm:
  - View data to be transmitted as a sequence of 16-bit integers.
  - Add the integers using 16 bit one's complement arithmetic.
  - Take the one's complement of the result – this result is the checksum
  - Receiver performs same calculation on received data and compares result with received checksum

Example

1 hexa digit needs 4bits

- Sender: IPV4 header in hexadecimal
  - 4500 0073 0000 4000 4011 c0a8 0001 c0a8 00c7 (16-bit words)
  - Sum up the words (can use 32 bits): 0002 479c
  - Add carry to the 16-bit sum: 479e
  - Take the complement: b861 → checksum

can detect 1 bit erros
but not bit

- Receiver:
  - Sum up the words including checksum (use 32 bits): 2fffd
  - Add carry to the 16 bit sum: ffff (= 0 in 1’s complement) → no error was detected

example values from wikipedia

Internet Checksum

- Not very strong in detecting errors
  - Pair of single-bit errors, one which increments a word, other decrements a word by same amount
- Why is it used still?
  - Very easy to implement in software
  - Majority of errors picked by CRC at link-level (implemented in hardware)

Cyclic Redundancy Check (CRC)

- Used by many link-level protocols: HDLC, DDCMP, Ethernet, Token-Ring
- Uses powerful math based on finite fields
- Background: Polynomial Arithmetic

Polynomial Arithmetic

- Represent a m bit message with a polynomial of degree “m-1”
  - 11000101 = $x^7 + x^6 + x^2 + 1$
- Arithmetic over the field of integers modulo 2 (coefficients are 1 or 0)
- Addition or subtraction are identical: XOR

X 0 1 1 1
Y 1 0 0 1

Z 1 1 1 0 (no carry..modulo2)

X + Y = Z

X 0 1 1 1
Z 1 1 1 0

Y 1 0 0 1

X +Z = Y
==> Z = Y - X

so additin subtarction are same

Polynomial Arithmetic

- Polynomial division (very similar to integer division)
  - X/Y is $X = q*Y + r$
  - For integers: $0<=r<Y$
  - For polynomials: degree of r (remainder polynomial) is less than divisor polynomial

Cyclic Redundancy Check (CRC)

- Message polynomial M(x): m bit message represented with a polynomial of degree “m-1”;
  - 11000101 = $x^7 + x^6 + x^2 + 1$
- Sender and receiver agree on a divisor polynomial C(x) of degree k
  - k: Number of redundancy bit
  - E.g. $C(x) = x^3+x^2+1$ (degree k = 3)
  - Choice of C(x) significantly effects error detection and is derived carefully based on observed error patterns
  - Ethernet uses CRC of 32 bits, HDLC, DDCMP use 16 bits
  - Ethernet: $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

Idea

- Sender sends m+k bits => Transmitted message P(x)
- Contrive to make P(x) exactly divisibly by C(x)
- Received message R(x)
  - No errors: R(x) = P(x), exactly divisible by C(x)
  - Errors: R(x) ~= P(x); likely not divisible by C(x)

Generate P(x)

- You have M(x) and C(x). Generate P(x)
- Multiply M(x) by x^k to get T(x)
  - Add k zeros at the end of the message
- Divide T(x) by C(x) to get remainder R(x)
- Subtract remainder R(x) from T(x) to get P(x)
- P(x) is now exactly divisible by C(x)

Details

- $T(x) = x^k M(x) = Q(x) C(x) + R(x)$
- $P(x) = x^k M(x) - R(x) = x^k M(x) + R(x) = Q(x)C(x)$
  - Coefficients of $R(x)$ are the redundant bits
  - Transmitted Bits: Messaged (n) bits, followed by redundant bits (k)

Example

POLYNOMAIL DIVISION.....Done usinmg MODULO 2 ARITHMETIC

So additiion, subtarction same...

- Message (M): 11001011
- Divisor (C): 1101
- T: 11001011000
- Remainder (R): 101
- Transmitted Bits (P): 11001011101

Error Detection

- Received polynomial = P(x) + E(x)

E(x) is the bits that are flipped

  - E(x) captures bit map of the positions of errors
- Cannot detect errors if E(x) is also divisible by C(x)
- Goal: Design C(x) such that for anticipated error patterns, E(x) is not divisible by C(x)

Example

- Detect all instances of odd number of bit errors
- E(x) contains odd number of terms with coefficient of ‘1’
  - Implies E(1) = 1

E(x) = C(x)Q(x)
put x = 1.

- If C(x) were a factor of E(x), then C(1) would also have to be 1
- If C(1) = 0, we can conclude C(x) does not divide E(x)
- If C(x) has some factor of the form x^i+1, then C(1)=0

C = x + 1 .....used on single bit parity i.e, even bit errro detection.....as degree = 1
so bit redundancy

Capabilities

- 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 (x^0 term)
- CRC is easily implementable on shift registers

Summary

- Important to detect errors in frames
- Many techniques exist (simple to complex)
  - Parity, Checksum, CRC
- Going Forward: Error Recovery