THE LINK LAYER

COMP 30650: NETWORKS AND INTERNET SYSTEMS

Dr. Gavin McArdle
Email: gavin.mcardle@ucd.ie
Office: A1.09 Computer Science

RECAP

Link Layer

- Framing
  - Byte/Bit stuffing
  - PPP on SONET
- Errors
  - Hamming Distance
    - Detection
    - Correction

TODAY'S PLAN

Error Detection

- Parity
- Checksum
- CRC

ERROR DETECTION

Some bits may be received in error due to noise. How do we detect this?

- Parity
- Checksums
- CRCs

Detection will let us fix the error, for example, by retransmission.

SIMPLE ERROR DETECTION – PARITY BIT

Take D data bits, add 1 check bit that is the sum of the D bits

- Sum is modulo 2 or XOR

SIMPLE ERROR DETECTION – PARITY BIT

Take D data bits, add 1 check bit that is the sum of the D bits

- Sum is modulo 2 or XOR

  1 0 0 1 1 0 0 _

SIMPLE ERROR DETECTION – PARITY BIT

Take D data bits, add 1 check bit that is the sum of the D bits

- Sum is modulo 2 or XOR

1 0 0 1 1 0 0 _

1 + 1 + 1 = 1 (3/2 = 1 with remainder is **1 (parity bit)**)

3%2 = 1 (parity bit)

PARITY BIT

How well does parity work?

- What is the distance of the code?
- How many errors will it detect/correct?
- What about larger errors?

PARITY BIT

How well does parity work?

- What is the distance of the code?
  - Flip 1 bit parity sum wrong
  - Flip another bit then we have another valid code
  - Distance is 2
- How many errors will it detect/correct?
  - Detect:
  - Correct:
- What about larger errors?

PARITY BIT

How well does parity work?

- What is the distance of the code?
  - Flip 1 bit parity sum wrong
  - Flip another bit then we have another valid code
  - Distance is 2
- How many errors will it detect/correct?
  - Detect: 1
  - Correct: 0
- What about larger errors?

PARITY BIT

How well does parity work?

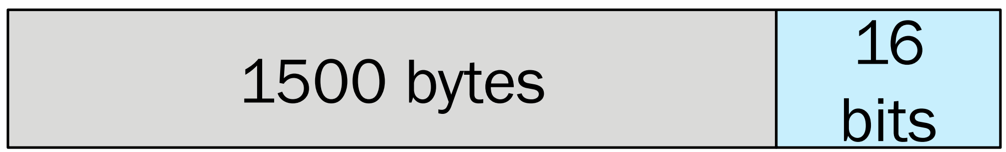
- What is the distance of the code?
  - Flip 1 bit parity sum wrong
  - Flip another bit then we have another valid code
  - Distance is 2
- How many errors will it detect/correct?
  - Detect: 1
  - Correct: 0
- What about larger errors?
  - Parity detects all odd number of errors

CHECKSUMS

Idea: sum up data in N-bit words

- Widely used in, e.g., TCP/IP/UDP

Stronger protection than parity



INTERNET CHECKSUM (2)

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

```
   0001
   f203
   f4f5
   f6f7
 +(0000)
 ------
  2ddf0

   ddf0
 +    2
 ------
   ddf2

   220d
```

INTERNET CHECKSUM

Sending:

1. Arrange data in 16-bit words

2. Put zero in checksum position and add

```
   0001
   f203
   f4f5
   f6f7
 +(0000)
 ------
  2ddf0
```

0 1 2 3 4 5 6 7 8 9 a b c d e f

INTERNET CHECKSUM

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position and add
3. Add any carryover back to get 16 bits

```
   0001
   f203
   f4f5
   f6f7
+(0000)
-------
  2ddf0

   ddf0
+     2
------
   ddf2
```

INTERNET CHECKSUM

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

```
    0001
    f203
    f4f5
    f6f7
+(0000)
------
  2ddf0

   ddf0
+     2
------
   ddf2

   220d
```

Negating:

 Ffff

-ddf2

- - - - -

220d

INTERNET CHECKSUM

Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
  0001
  f203
  f4f5
  f6f7
+ 220d
------
 2fffd

  fffd
+    2
------
  ffff

  0000
```

INTERNET CHECKSUM

Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
   0001
   f203
   f4f5
   f6f7
+  220d
------
  2fffd

   fffd
+     2
------
   ffff

   0000
```

INTERNET CHECKSUM

How well does the checksum work?

- What is the distance of the code?
  - Two corresponding errors could fool this checksum (e.g. add 16 and remove 16) sum is ok and it is valid codeword
  - 2
- How many errors will it detect/correct?
  - Detect: 1
  - Correct: 0

What about larger errors?

- This is where its better
  - Will find all burst errors up to 16
  - Burst – sequence/window of errors in a row.
  - Errors of 16 or less will be detected

CYCLIC REDUNDANCY CHECK (CRC)

Even stronger protection

- Given n data bits, generate k check bits such that the n+k bits are **evenly** divisible by a **generator C**

Example with numbers:

- n = 302, k = one digit, C = 3
- 3 0 2 _   should be divisible by 3 evenly
- 3 0 2 0 / 3 = remainder is 2 -> 1 short of even divisible
- 3 0 2 1 is evenly divisible

CRCS

Send Procedure:

1. Extend the n data bits with k zeros
2. Divide by the generator value C
3. Keep remainder, ignore quotient
4. Adjust k check bits by remainder

Receive Procedure:

1. Divide and check for zero remainder

CRCS

Data bits: 1101011111

Check bits:
$C(x)=x^4+x^1+1$

C = 10011

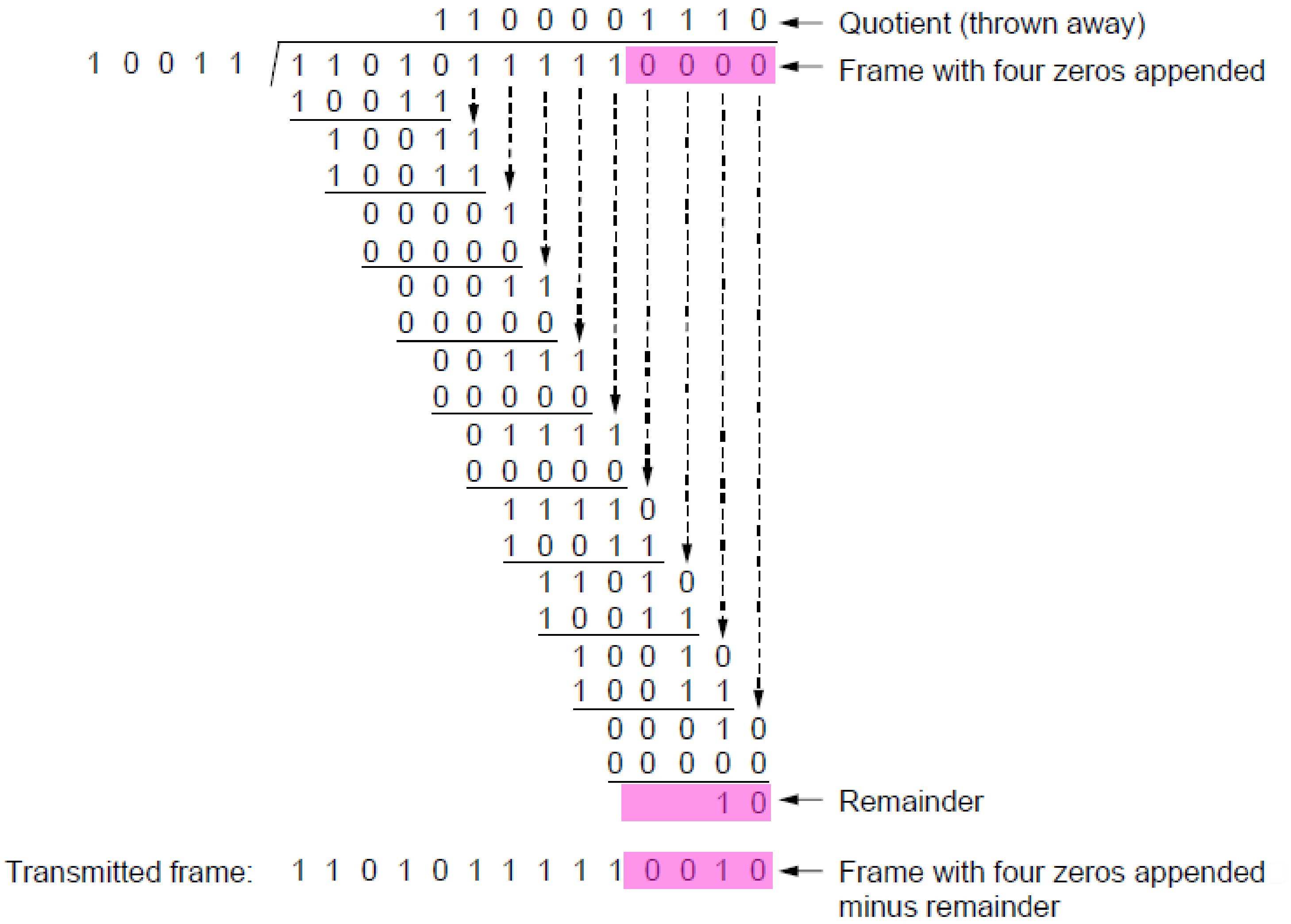
k = 4

$$1\,0\,0\,1\,1 \overline{)\,1\;1\;0\;1\;0\;1\;1\;1\;1\;1\,}$$

USE XOR NOT SUBTRACTRION!

0 XOR 0 -> 0
0 XOR 1 -> 1
1 XOR 1 -> 0
1 XOR 0 -> 1

CRCS



1 1 0 0 0 0 1 1 1 0
Quotient (thrown away)
1 0 0 1 1
1 1 0 1 0 1 1 1 1 1 0 0 0 0
Frame with four zeros appended
1 0 0 1 1
1 0 0 1 1
1 0 0 1 1
0 0 0 0 1
0 0 0 0 0
0 0 0 1 1
0 0 0 0 0
0 0 1 1 1
0 0 0 0 0
0 1 1 1 1
0 0 0 0 0
1 1 1 1 0
1 0 0 1 1
1 1 0 1 0
1 0 0 1 1
1 0 0 1 0
1 0 0 1 1
0 0 0 1 0
0 0 0 0 0
1 0
Remainder
Transmitted frame:
1 1 0 1 0 1 1 1 1 1 0 0 1 0
Frame with four zeros appended minus remainder

CRCS

Protection depend on generator

- Standard CRC-32 is 1 0000 0100 1100 0001 0001 1101 1011 0111

Properties:

- HD=4, detects up to triple bit errors
- Also odd number of errors
- And bursts of up to k bits in error
- Not vulnerable to systematic errors like checksums
  - Eg adding zeros to a sum cause no errors.

ERROR DETECTION IN PRACTICE

CRCs are widely used on links

- Ethernet, 802.11, ADSL, Cable …

Checksum used in Internet

- IP, TCP, UDP … but it is weak

Parity

- Is little used