Section 2 – Link Layer

CSE 461 – Autumn 2015

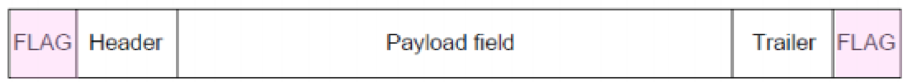
Panji Wisesa

Byte Count

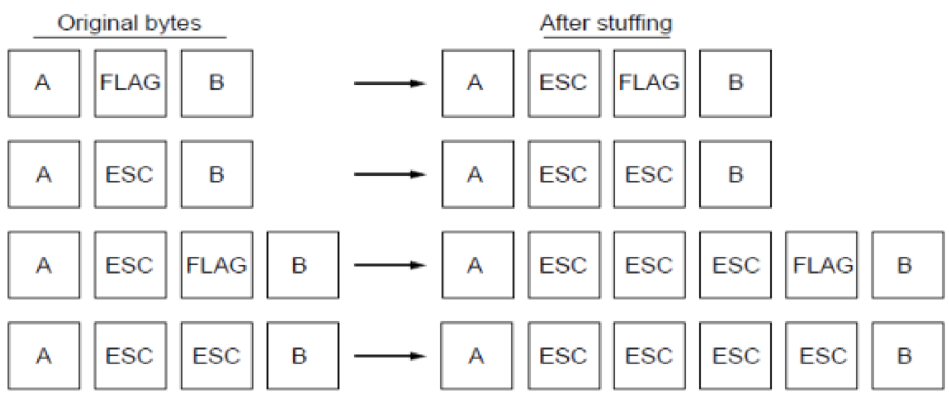
- Add a length to the start if the frame
- No protection against any errors

Byte Stuffing

- Have a special flag byte value that means start/end of frame

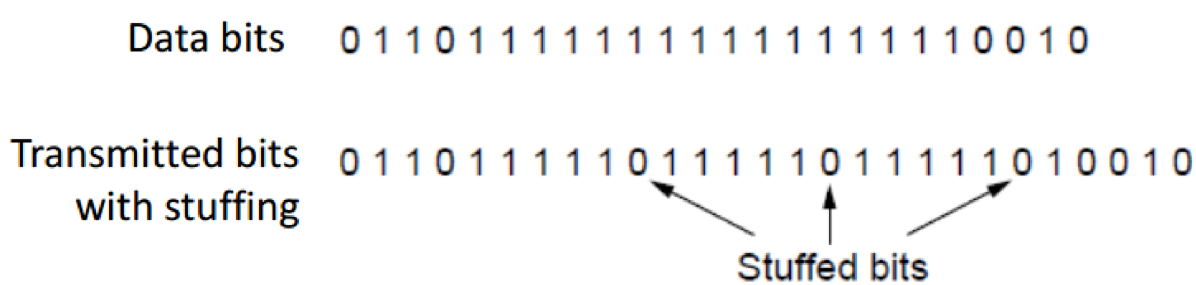


- Replace the flag inside the frame with an escape code



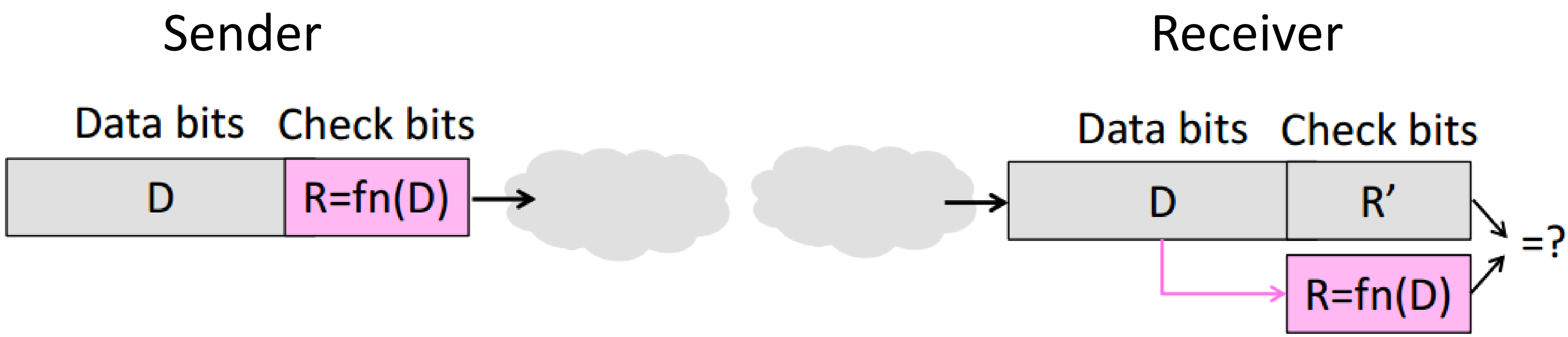
Bit Stuffing

- Like byte stuffing but in the bit level
- Use six consecutive 1s as the flag
  - On transmit, after five 1s in the data, insert a 0
  - On receive, a 0 after five 1s is deleted



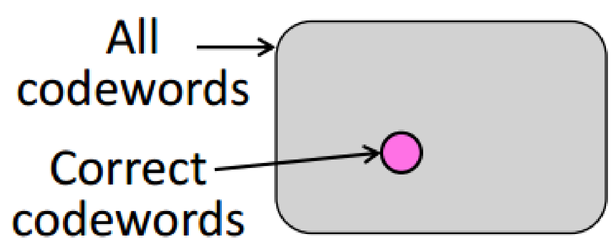
Error Detection and Correction

- Done with check bits, calculated from the data to be transmitted
- More check bits usually means more errors can be detected and calculated
- However, it's a balance between the overhead of check bits and the reliability from those check bits



Why Check Bits Work

- The combination of the data and check bits can be called a codeword
- The check bit works because there's a lot more codewords than valid ones (the check bits matches the check bits calculated from the data)
- So it's very unlikely that errors can transform a valid codeword into a different valid codeword



Hamming Distance

- Distance is the number of bit flips needed to change D1 to D2
- Hamming distance of a code is the minimum distance between any pair of valid codewords
- For a code of distance d+1, up to d errors will always be detected
- For a code of distance 2d+1, up to d errors can always be corrected by mapping to the closest codeword

Error Detection

- Standard functions to create the check bits:
  - Parity bit, 1 check bit from the sum of all data bits, Hamming distance of 2
  - Checksum, 16 check bits from 16-bit ones complement arithmetic, Hamming distance of 2, good for Burst Errors
  - CRC (Cyclic Redundancy Check), k check bits from n data bits such that n+k bits are evenly divisible by a generator C, Hamming distance of 4, good for Burst Errors up to k bits

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
```

Receiving:

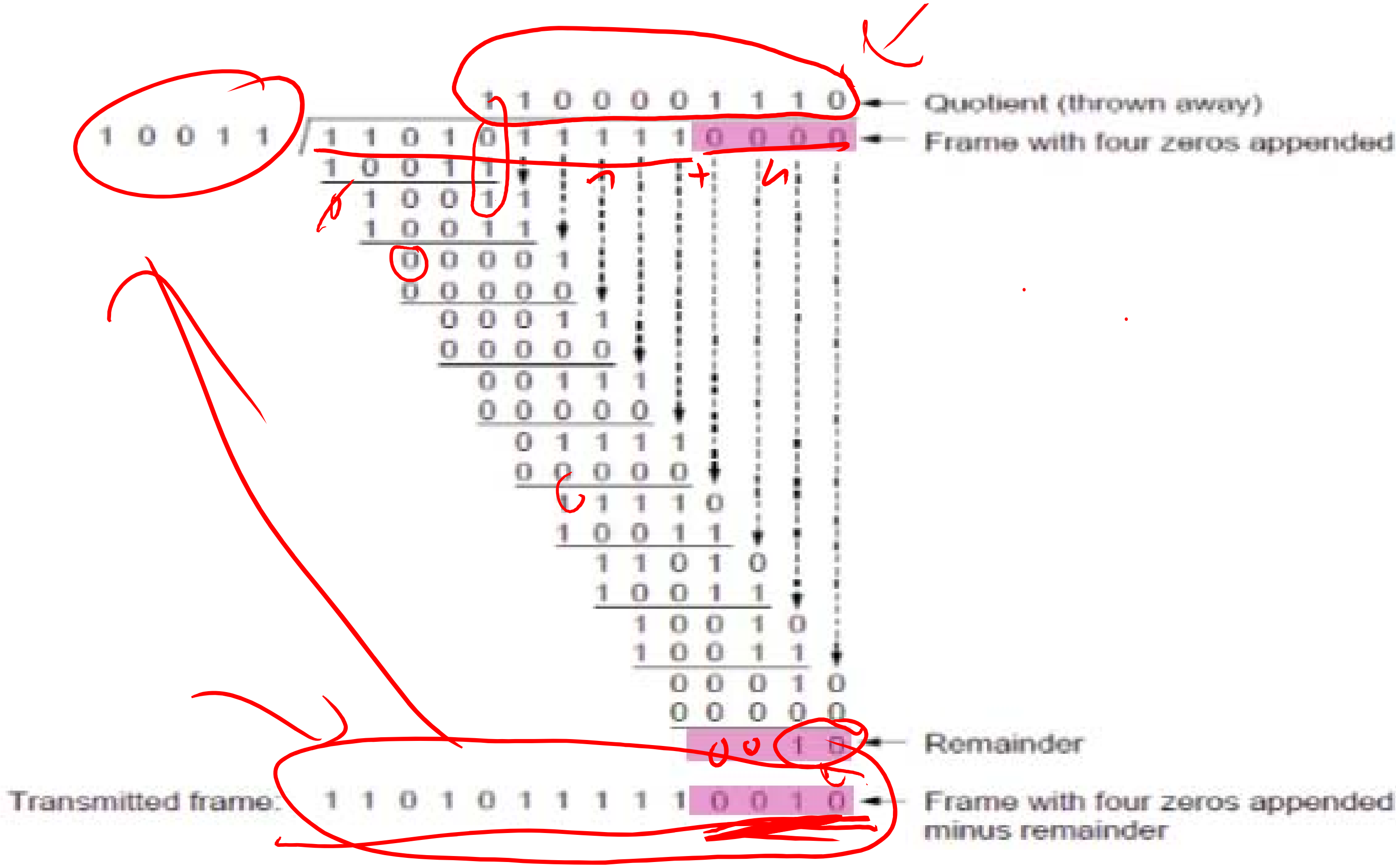
1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
   0001
   f203
   f4f5
   f6f7
+  220d
------
  2fffd
    ↓
   fffd
+     2
------
   ffff
    ↓
   0000
```

CRC

Data bits:
1101011111

Check bits:
$C(x)=x^4+x^1+1$
C = 10011
k = 4



Error Correction

- Harder than detection, can correct only d errors in codewords with Hamming distance >= 2d +1
- In this class we will mostly talk about Hamming Code for error correction

Hamming Code

- Allows the creation of a codeword with a Haming distance of 3, for every n data bits there must be k check bits where ($n = 2^k – k – 1$)
- The check bits are located in positions that are powers of 2, so $1 = 2^0$, $2 = 2^1$, $4 = 2^2$, etc.
- Check bits in position p is parity for positions with a p term in their values

Hamming Code Check Bits Coverage

Data = 4 bits, Check bits = 3 bits, Codeword = 7 bits

Check bits are located at:

- 1 = 2^0, which means they cover 3, 5, & 7
- 2 = 2^1, which means they cover 3, 6, & 7
- 4 = 2^2, which means they cover 5, 6, & 7

What the check bits cover are determined by whether the location contains them in their term or in other words, the location in binary has a 1 at the check bit's power to 2.

The value of the check bits themselves are the summation of the bits at those positions.

Decimal	Binary
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Hamming Code Example

To decode:

- Recompute check bits (with parity sum including the check bit)
- Arrange as a binary number
- Value (syndrome) tells error position
- Value of zero means no error
- Otherwise, flip bit to correct

0	1	0	0	1	1	1
1	2	3	4	5	6	7

p_1= 0+0+1+1 = 0, p_2= 1+0+1+1 = 1,
p_4= 0+1+1+1 = 1

Syndrome = 1 1 0, flip position 6
Data = 0 1 0 1 (correct after flip!)

Error Detection vs. Correction

- Usually error correction is used when errors are expected and there’s no time to retransmit
- While error detection is more efficient when errors are not expected or when the errors are really large so no hope of correction anyway
- But to choose one or the other still depends on the amount of data being sent and the rate of error