6.263/16.37: Lecture 2

The Data Link Layer: Framing and Error Detection

Eytan Modiano
MIT

Data Link Layer (DLC)

- **Responsible for reliable transmission of packets over a link**
  - **Framing: Determine the start and end of packets (sec 2.5)**
  - **Error Detection: Determine when a packet contains errors (sec 2.3)**
  - **Error recovery: Retransmission of packets containing errors (sec 2..4)**

    DLC layer recovery

    May be done at higher layer

Framing

010100111010100100101010100111000100

Where is the DATA??

- **Three approaches to find frame and idle fill boundaries:**

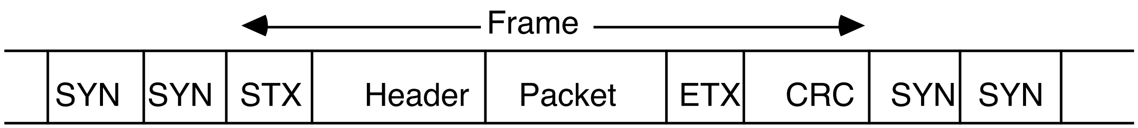
  1) Character oriented framing

  2) Length counts

  - fixed length

  3) Bit oriented protocols (flags)

Character Based Framing



SYN is synchronous idle
STX is start text
ETX is end text

- **Standard character codes such as ASCII and EBCDIC contain special communication characters that cannot appear in data**

- **Entire transmission is based on a character code**

Issues With Character Based Framing

- Character code dependent
  - How do you send binary data?
- Frames must be integer number of characters
- Errors in control characters are messy

NOTE: Primary Framing method from 1960 to ~1975

Length field approach (DECNET)

- Use a header field to give the length of the frame (in bits or bytes)
  - Receiver can count until the end of the frame to find the start of the next frame
  - Receiver looks at the respective length field in the next packet header to find that packet's length
- Length field must be log2 (Max_Size_Packet) + 1 bits long
  - This restricts the packet size to be used
- Issues with length counts
  - Difficult to recover from errors
  - Resynchronization is needed after an error in the length count

Fixed Length Packets (e.g., ATM)

- All packets are of the same size
  - In ATM networks all packets are 53 Bytes
- Requires synchronization upon initialization
- Issues:
  - Message lengths are not multiples of packet size
    - Last packet of a message must contain idle fill (efficiency)
  - Synchronization issues
  - Fragmentation and re-assembly is complicated at high rates

Bit Oriented Framing (Flags)

- **A flag is some fixed string of bits to indicate the start and end of a packet**
  - **A single flag can be used to indicate both the start and the end of a packet**

- **In principle, any string could be used, but appearance of flag must be prevented somehow in data**
  - **Standard protocols use the 8-bit string 01111110 as a flag**
  - **Use 01111111..1110 (<16 bits) as abort under error conditions**
  - **Constant flags or 1's is considered an idle state**

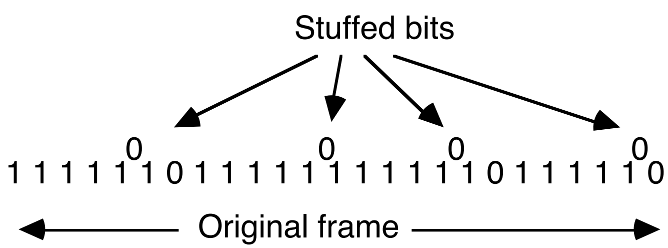
- **Thus 0111111 is the actual bit string that must not appear in data**

- **INVENTED ~ 1970 by IBM for SDLC (synchronous data link protocol)**

BIT STUFFING (Transmitter)

- **Used to remove flag from original data**

- **A 0 is stuffed after each consecutive five 1's in the original frame**



- **Why is it necessary to stuff a 0 in 0111110?**
  - **If not, then**

    0111110111 -> 0111110111
    011111111 -> 0111110111

  - **How do you differentiate at the receiver?**

DESTUFFING (Receiver)

- **If 0 is preceded by 011111 in bit stream, remove it**
- **If 0 is preceded by 0111111, it is the final bit of the flag.**

Example: Bits to be removed are underlined below

$10011111\underline{0}1100111011111\underline{0}110\underbrace{\underline{01111110}}_{\text{flag}}$

Overhead

- In general with a flag 01^K0 the bit stuffing is require whenever 01^{k-1} appears in the original data stream
- For a packet of length L this will happen about $L/2^k$ times

$$E\{OH\} = L/2^k + (k+2) \text{ bits}$$

- For 8 bit flag OH ~ 8 + L/64
  - For large packets efficiency ~ 1 - 1/64 = 98.5 (or 1.5% overhead)
- Optimal flag length
  - If packets are long want longer flag (less stuffing)
  - If packets are short want short flag (reduce overhead due to flag)

$$K_{opt} \sim \log_2(L)$$

Framing Errors

- All framing techniques are sensitive to errors
  - An error in a length count field causes the frame to be terminated at the wrong point (and makes it tricky to find the beginning of the next frame)
  - An error in DLE, STX, or ETX causes the same problems
  - An error in a flag, or a flag created by an error causes a frame to disappear or an extra frame to appear
- Flag approach is least sensitive to errors because a flag will eventually appear again to indicate the end of a next packet
  - Only thing that happens is that an erroneous packet was created
  - This erroneous packet can be removed through an error detection technique

Error detection techniques

- **Used by the receiver to determine if a packet contains errors**
- **If a packet is found to contain errors the receiver requests the transmitter to re-send the packet**

- **Error detection techniques**

  - **Parity check**
    - **single bit**
    - **Horizontal and vertical redundancy check**

  - **Cyclic redundancy check (CRC)**

Effectiveness of error detection technique

- Effectiveness of a code for error detection is usually measured by three parameters:

 1) minimum distance of code (d) (min # bit errors undetected)
 The minimum distance of a code is the smallest number of errors that can map one codeword onto another. If fewer than d errors occur they will always detected. Even more than d errors will often be detected (but not always!)

 2) burst detecting ability (B) (max burst length always detected)

 3) probability of random bit pattern mistaken as error free (good estimate if # errors in a frame >> d or B)
 – Useful when framing is lost
 – K info bits => 2^k valid codewords
 – With r check bits the probability that a random string of length k+r maps onto one of the 2^k valid codewords is $2^k/2^{k+r} = 2^{-r}$

Parity check codes

k Data bits	r Check bits

- **Each parity check is a modulo 2 sum of some of the data bits**

Example:

$$c_1 = x_1 + x_2 + x_3$$
$$c_2 = x_2 + x_3 + x_4$$
$$c_3 = x_1 + x_2 + x_4$$

Single Parity Check Code

- **The check bit is 1 if frame contains odd number of 1's; otherwise it is 0**

  1011011 -> 1011011 1
  1100110 -> 1100110 0

- **Thus, encoded frame contains even number of 1's**
- **Receiver counts number of ones in frame**
  - **An even number of 1's is interpreted as no errors**
  - **An odd number of 1's means that an error must have occured**
    - **A single error (or an odd number of errors) can be detected**
    - **An even number of errors cannot be detected**
    - **Nothing can be corrected**

- **Probability of undetected error (independent errors)**

$$P(undetected) = \sum_{i\ even} \binom{N}{i} p^{i}(1-p)^{N-i}$$

N = packet size
p = error prob.

Horizontal and Vertical Parity

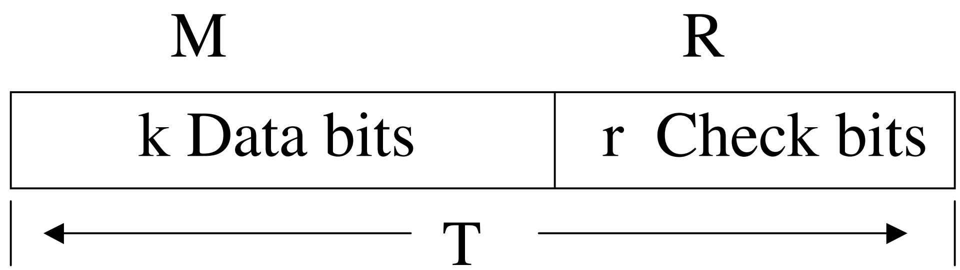
1	0	0	1	0	1	0	1	
0	1	1	1	0	1	0	0	Horizontal
1	1	1	0	0	0	1	0	checks
1	0	0	0	1	1	1	0	
0	0	1	1	0	0	1	1	
1	0	1	1	1	1	1	0	

Vertical checks

1	0	0	1	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	0	1	1	1	0
0	0	1	1	0	0	1	1
1	0	1	1	1	1	1	0

- **The data is viewed as a rectangular array (i.e., a sequence of words)**
- **Minimum distance=4, any 4 errors in a rectangular configuration is undetectable**

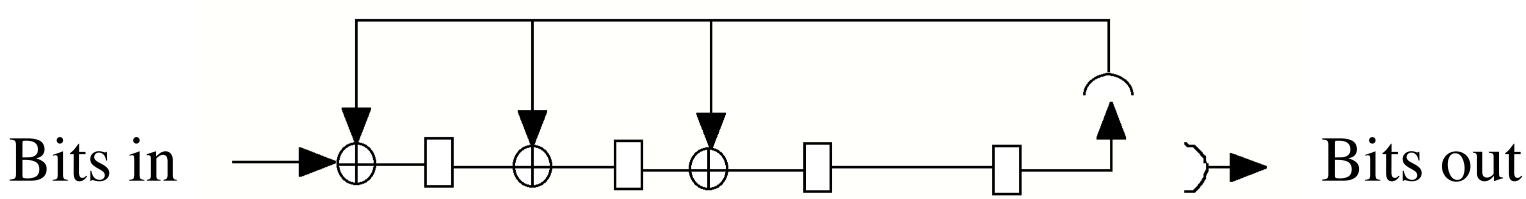
Cyclic Redundancy Checks (CRC)



M = info bits
R = check bits
T = codeword

$$T = M\,2^r + R$$

- **A CRC is implemented using a feedback shift register**



Cyclic redundancy checks

$$T = M\,2^r + R$$

- **How do we compute R (the check bits)?**
  - **Choose a generator string G of length r+1 bits**
  - **Choose R such that T is a multiple of G (T = A*G, for some A)**
  - **Now when T is divided by G there will be no remainder => no errors**
  - **All done using mod 2 arithmetic**

$T = M\,2^r + R = A*G \Rightarrow M\,2^r = A*G + R$ (mod 2 arithmetic)

Let R = remainder of $M\,2^r/G$ and T will be a multiple of G

- **Choice of G is a critical parameter for the performance of a CRC**

Example

r = 3, G = 1001

M = 110101 => $M2^r$ = 110101000

```
          110011
        ________________
1001   | 110101000
         1001
         01000
          1001
          0001100
             1001
             01010
              1001
               011 = R (3 bits)
```

Modulo 2
Division

Checking for errors

- **Let T' be the received sequence**
- **Divide T' by G**
  - **If remainder = 0 assume no errors**
  - **If remainder is non zero errors must have occurred**

Example:
Send T = 110101011
Receive T' = 110101011
(no errors)

No way of knowing how many errors occurred or which bits are In error

```
1001 | 110101011
       1001
       ----
       01000
        1001
        ----
        0001101
             1001
             ----
             01001
              1001
              ----
               000 => No errors
```

Mod 2 division as polynomial division

Implementing a CRC

Performance of CRC

- For r check bits per frame and a frame length less than 2^{r-1}, the following can be detected

    1) All patterns of 1,2, or 3 errors (d > 3)
    2) All bursts of errors of r or fewer bits
    3) Random large numbers of errors with prob. $1-2^{-r}$

- Standard DLC's use a CRC with r=16 with option of r=32

    – CRC-16, $G = X^{16} + X^{15} + X^2 + 1 = 11000000000000101$

Internet Checksum

- **In the internet, error detection is done at different layers: IP, TCP**
  - **Implemented in software**
  - **CRC is too complex for software implementation**

- **The internet uses a 1's complement sum instead of a CRC**
  - **Very easily implemented in software**
  - **Reasonable performance**

- **One's complement sum**
  - **Mod-2 addition with carry-out; a carry-out in the most-significant -bit is wrapped around and added to the least-significant bit**
  - **Take one's complement of "one's complement sum"**

- **Reasonable performance, but:**
  - **Minimum distance is 2 (rare) - try to find it**
  - **Can you construct a 3 bit error pattern that cannot be detected?**

$$
\begin{array}{rr}
 & \mathbf{1010011} \\
 & \underline{\mathbf{0110110}} \\
\text{carry-out} \;\; 1 & \mathbf{0001001} \\
\text{Carry wrap-around} & \underline{\mathbf{0000001}} \\
 & \mathbf{0001010} \\
\textbf{One's complement} = & \mathbf{1110101}
\end{array}
$$

Physical Layer Error Characteristics

- Most Physical Layers (communications channels) are not well described by a simple BER parameter
- Most physical error processes tend to create a mix of random & bursts of errors
- A channel with a BER of 10^{-7} and a average burst size of 1000 bits is very different from one with independent random errors

- Example: For an average frame length of 10^4 bits
  - random channel: E[Frame error rate] ~ 10^{-3}
  - burst channel: E[Frame error rate] ~ 10^{-6}

- Best to characterize a channel by its Frame Error Rate

- This is a difficult problem for real systems