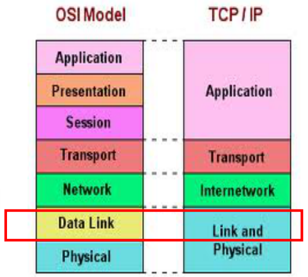
Dr SABER Takfarinas
takfarinas.saber@ucd.ie

COMP2009J
Computer Networks

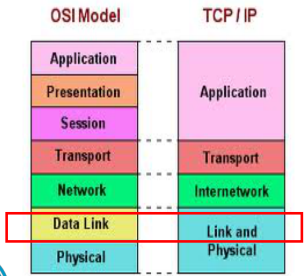
Link Layer

Last Session



- How to send messages/packets in the same LAN
- Using ARP Tables

This Session



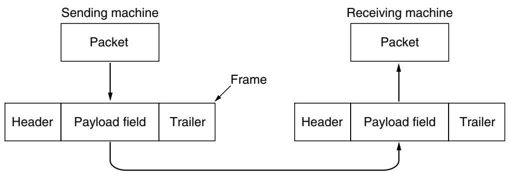
- What are these messages/packets made of?
- What do they contain?

The Link Layer

- In the physical layer we looked at how digital information is transferred between two machines
- In the link layer we are concerned with how to send messages between two machines
  - Messages are called **frames**
- The link layer has a number of functions, some important ones are
  - Framing messages
  - Dealing with transmission errors
  - Regulating the flow of data so slow receivers are not swamped

1- Frames

- So far we have talked about sending packets, in the link layer these are put inside another structure called a frame
- Frames have additional information added in headers (before the message) and trailers (after the message)

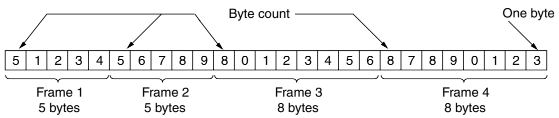


Framing

- Breaking up the bit stream into frames is more difficult than it at first appears
- A good design must make it easy for a receiver to find the start of new frames while using little of the channel bandwidth
- We will look at three methods:
  - Byte count
  - Flag bytes with byte stuffing
  - Flag bits with bit stuffing

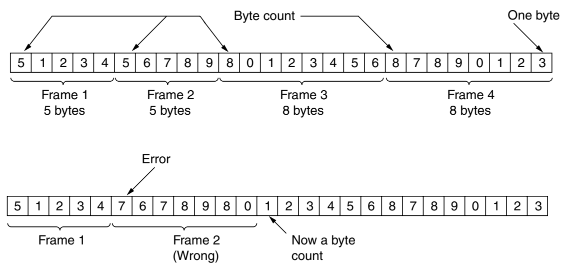
Framing - Byte Count

- One of the simplest and most efficient methods
- Simply start the frame by having the first number be the count of how many bytes are in the frame



Framing - Byte Count

- But what happens if the byte is corrupted in transmission?



Framing – Flag Byte

- The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes
- Often the same byte, called a **flag byte**, is used as both the starting and ending delimiter
- Two consecutive flag bytes indicate the end of one frame and the start of the next
- Thus, if the receiver ever loses synchronization it can just search for two flag bytes to find the end of the current frame and the start of the next frame

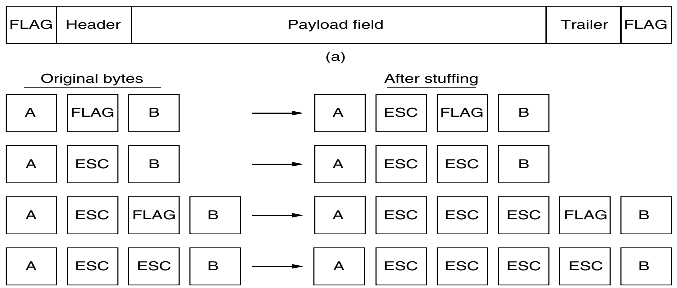
Framing – Flag Byte

- What happens if the flag byte appears in the data being sent?
- One solution is to insert a special **escape byte** (ESC) before any accidental flag byte
- This way we can tell when a frame ends or the byte is just data
- The receiver then removes these flag and escape bytes from data it receives
- This process is called **byte stuffing**

Framing – Escape Byte

- What happens if the escape byte appears in the data being sent?
- The simple solution is that another escape byte is stuffed before it

Framing - Flag Byte



Framing – Flag Bits

- Byte stuffing has a drawback, adding these bytes increases the size of the message a lot
  - Because each flag adds another byte
- The third method of framing uses single bits for stuffing so the increase in size is not as much

Framing – Flag Bits

- Each frame begins with the bit pattern 0$\underbrace{111111}_{\text{six 1s}}$0
- Bit Stuffing
  - **Sender:** Whenever we see five 1s in our data we **insert** a 0
  - **Receiver:** Whenever we see five 1s we **remove** the following 0
- It is the same idea as byte stuffing but with less overhead
- The only problem is that the final frame could be of any **number of bits** but byte stuffing it is always a **number of bytes**

Framing – Flag Bits

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

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

Stuffed bits

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

Framing – Flag Bits

$\boxed{0\ 1\ 1\ 0\ 1\ 1\ 1\ 1}\ \boxed{1\ 1\ 1\ 1\ 1\ 1\ 1\ 1}\ \boxed{1\ 1\ 1\ 1\ 0\ 0\ 1\ 0}$

$\boxed{0\ 1\ 1\ 0\ 1\ 1\ 1\ 1}\ \boxed{1\ 0\ 1\ 1\ 1\ 1\ 1\ 0}\ \boxed{1\ 1\ 1\ 1\ 1\ 0\ 1\ 0}\ \boxed{0\ 1\ 0}$

Stuffed bits

Physical Layer Framing

- There exists other methods of framing that we are not studying closely
- Some of them exploit knowledge of what is happening in the physical layer
  - If the physical layer is using a protocol such as 4B/5B
  - It means that we know that **certain byte sequences** cannot appear in the data
  - Therefore, we can use one of these Byte sequences to signal the start and end of frames

2- Errors

- What the physical layer does is accept a raw bit stream and attempt to deliver it to the destination
- If the channel is noisy, the physical layer will add some redundancy to its signals to reduce the bit error rate
- However, the bit stream received by the data link layer is not guaranteed to be error free
- Some bits may have different values and the number of bits received may differ from the number of bits transmitted
- It is up to the data link layer to **detect** and, if necessary, **correct** errors

Detecting Errors

- The usual approach is for the data link layer to break up the bit stream into discrete frames, compute a short token called a checksum for each frame, and include the checksum in the frame when it is transmitted
- When a frame arrives at the destination, the checksum is recomputed
- If the newly computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it

Error Control

- As we know when a signal is transmitted by the physical layer attenuation and distortion can cause the signal to be read incorrectly
- There are a number of things we can do
  - Use codes to detect if errors have occurred
  - Use codes to correct errors when they occur
  - Retransmit any lost frames
- Reliability is a concern that appears in multiple layers

Adding Redundancy

- If we add extra information to the frame we can use these to help detect or correct errors
- Error Detection
  - Add **check bits** to messages that let some errors be detected
- Error Correction
  - Add more **check bits** to messages that let some errors be corrected
- The hardest problem is to structure the codes to detect as many errors with a **small amount** of extra bits and not too much calculation

Simple Error Detection Example

- A simple code for detecting errors
  - Send two copies, if they are different there has been an error
- How good is this code?
  - How many errors can it detect/correct?
  - How many errors will make it fail?
- How much overhead does it add?

Using Error Codes

- Codeword consists of D data bits plus R check bits
  - Called systematic block code
- Number of check bits depends on the size of the data
- Check bits are computed based on the data and then added to the end
- When the receiver gets the package it recomputes the check bits based on the data bits
  - If there are no errors the check bits should match
- When an error is detected it can be difficult to tell if the error is in the data or in the check bits

Single Parity Check

- Use a single check bit such that the number of 1s in every codeword is **even**
  - If the number of 1s in the data is even we add a 0
  - If the number of 1s in the data is odd we add a 1
- This is called **single parity check**
- If receiver gets a codeword with an odd number of 1's, it knows error(s) occurred
  - can **detect** any odd number of bit errors
  - but: can't tell how many errors, or which bits are in error
  - even worse: any even number of bit errors is *undetectable*

Single Parity Check - Example

- Data to be transmitted: 10110101
  - There are 5 1s in the data so the parity bit is 1
- We transmit: 101101011
  - If receiver gets 101101011 parity check is ok
  - If receiver gets 101100011 parity check fails
  - If receiver gets 101110011 parity check is ok but codeword is incorrect
  - If receiver gets 001100011 parity check is ok but codeword is incorrect
- Data to be transmitted: 10110001
  - There are 4 1s in the data so the parity bit is 0

2 Dimensional Parity Check

- In this scheme we form data into a 2-dimensional array and add single parity check bits to each row and each column
- If we have the data 1110001 1000111 0011001
- We form a 3x7 array and add row and column parity bits
  - 1 1 1 0 0 0 1 **0**
  - 1 0 0 0 1 1 1 **0**
  - 0 0 1 1 0 0 1 **1**
  - **0 1 0 1 1 1 1 1**

2 Dimensional Parity Check

- The receiver knows to form received bit string into 4x8 array, then check the row and column parity bits
- This scheme can **detect** any odd number of bit errors in a row or column
  - It can also detect an even number of bit errors if they're in a single row (using the column parity checks)
  - It can also detect an even number of bit errors if they're in a single column (using the row parity checks)
- This scheme can **correct** any single bit error

Example 1 - 1 Bit Error

- If we have the data 1110001 1000111 0011001
- And the receiver gets
  - 1 1 0 0 0 0 1 **0**
  - 1 0 0 0 1 1 1 **0**
  - 0 0 1 1 0 0 1 **1**
  - **0 1 0 1 1 1 1 1**
- The receiver can detect that the bit in position (1, 3) was in error and can correct it

Example 2 - 2 Bit Errors

- If we have the data 1110001 1000111 0011001
- And the receiver gets
  - 0 1 0 0 0 0 1 0
  - 1 0 0 0 1 1 1 0
  - 0 0 1 1 0 0 1 1
  - 0 1 0 1 1 1 1 1
- The receiver can detect that the bit errors have occurred but it cannot correct them

Example 3 - 2 Bit Errors

- If we have the data 1110001 1000111 0011001
- And the receiver gets
  - 0 1 1 0 0 0 1 0
  - 1 0 1 0 1 1 1 0
  - 0 0 1 1 0 0 1 1
  - 0 1 0 1 1 1 1 1
- The receiver can detect that the bit errors have occurred but it cannot correct them
- The same parity checks would be received if the errors were in positions (1, 3) and (2, 1)

Example 4 - 3 Bit Errors

- If we have the data 1110001 1000111 0011001
- And the receiver gets
  - 0 0 0 0 0 0 1 0
  - 1 0 0 0 1 1 1 0
  - 0 0 1 1 0 0 1 1
  - 0 1 0 1 1 1 1 1
- The receiver can detect that the bit errors have occurred but it cannot correct them
- Two of the bit errors could be on a single row with a correct parity so we cannot correct them

Example 5 - 4 Bit Errors

- If we have the data 1110001 1000111 0011001
- And the receiver gets
  - 0 1 0 0 0 0 1 0
  - 1 0 1 0 1 1 1 0
  - 0 0 1 1 0 0 1 1
  - 1 1 0 1 1 1 1 1
- The receiver can detect that the bit errors have occurred but cannot correct them
- Errors in (1,1), (1,3), (2,1) and (4,3) would have the same parity check