# The Data Link Layer

CS158a

Chris Pollett

Feb 26, 2007.

# Outline

- Finish up Overview of Data Link Layer
- Error Detecting and Correcting Codes

# Finish up Overview of Data Link Layer

- Last day we were explaining different techniques for handling framing in the data link layer.
- The other two important things done in the data link layer are error detection/correction and flow control.
- Before we go into detail about error detection/correction, we briefly mention that there are two main techniques for flow control:
  - **feedback based**, where the receiver sends back information to the sender giving its permission to send more data
  - **rate-based**, where the protocol has a built in mechanism to limit the rate at which the sender can send without needing feedback.
- In the data link layer only the first approach is used, and we will consider various sliding window protocols that say how this feedback is done.
- But first, …

# Error Correcting Codes

- One goal of the data link layer is to provide an error free channel for the network layer.
- To do this it must be able to deal with errors in the data it receives from the physical layer.
- The two most common strategies to deal with errors are:
  - to use an **error detecting code** to determine in a block of bits whether any of the bits were incorrectly transmitted. If so, one requests the block be re-sent. This method is typically used on highly reliable channels where one doesn't have to retransmit that often
  - to use an **error correcting code** to determine errors in blocks and correct them. This might involve more redundant bits being sent but might work better on a noisy channel.

# The Set-up

- We will assume a frame consists of m bits of data, r redundant bits, and has a total length is n = m+r.
- An n-bit unit containing data and checks bits is referred to as an n-bit **codeword**.
- The number of bit positions at which the two codewords differ is called their **Hamming distance**. For example, given the two codewords 10**001**001 and 10**110**001, their Hamming distance is 3, as the bit positions colored blue differ.
- To detect  up to d errors you need each of your codewords to be of Hamming distance at least d+1 apart.
- To correct up to d errors you need each of your codewords to be at least 2d+1 apart.

# Example Codes

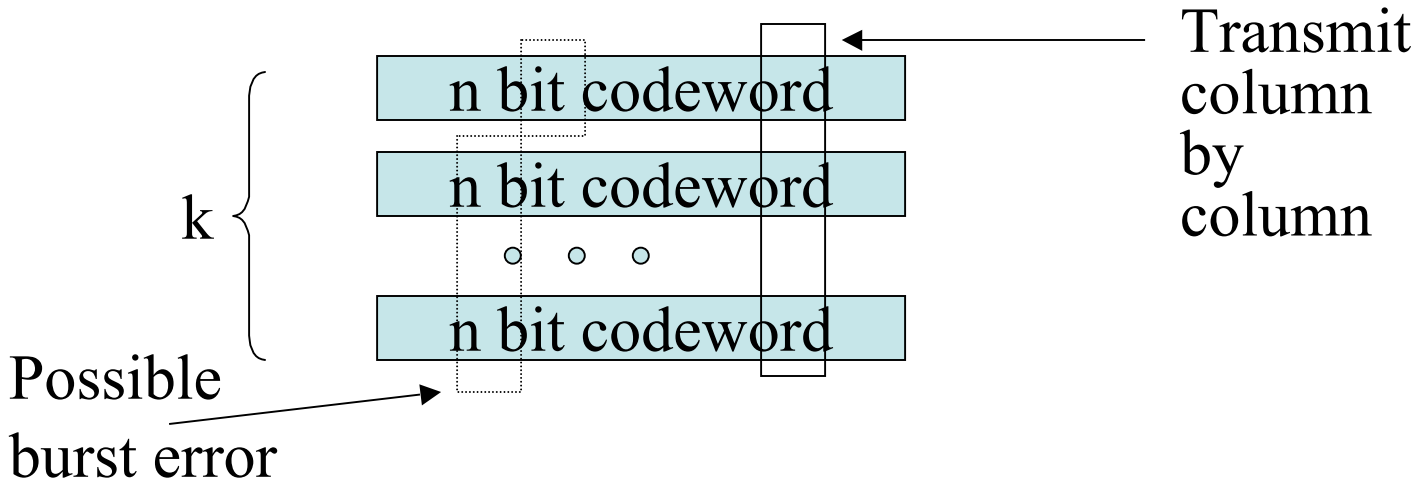
- One simple error detecting code is to add to your m bit message a single bit (called a **parity bit**) which contains the sum of the other bits mod 2. This can detect single bit errors in the n bits that are transmitted. So if your message was 0101, you would send 01010; if your message 1101, you would send 11011.
- As an example error correcting code consider the codewords: 0000000000, 0000011111, 1111100000, 1111111111. These are distance 5 apart. If any two bits are flipped in any codeword, one codeword is still closer to that code than any other so we can correct the error. For instance, if you receive 0000011000, you would decode this as 0000000000 since it is nearest to this codeword among the four possibilities.

# Hamming Codes

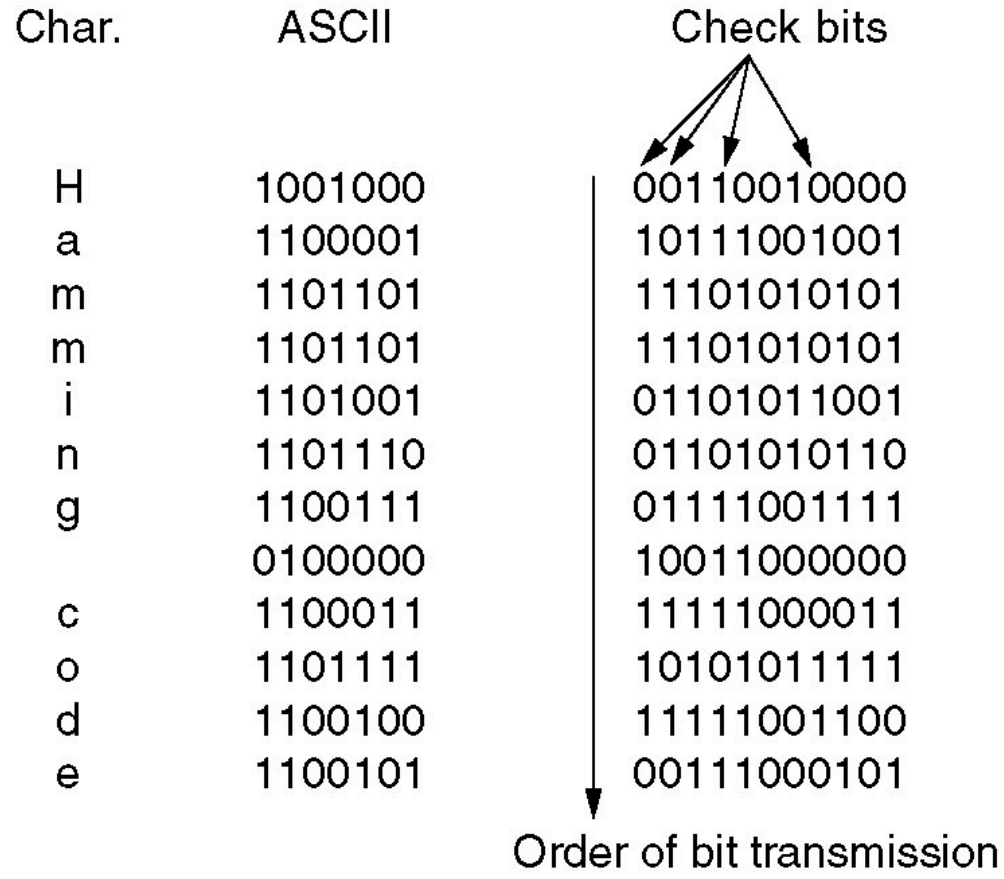
- Suppose we want to correct any single bit error
- There are $2^m$ legal messages of length m.
- In an n bit code, each of these legal messages has n illegal neighbors at distance 1.
- So to correct single errors we need that $(n+1)2^m \leq 2^n$. i.e., that $(m+r+1) \leq 2^r$.
- A Hamming code achieves this bound.
- Bit locations are numbered from 1 to n. Those that are powers of 2 are check bits (parities of all bits to the left), the remaining bit locations are used for the message.
- As an example, if the code word was **00**1**1**001**0**000, the blue bits are check bits. The first data bit is at location 3. Since the binary expansion of 3 is $2^1+2^0$. The 1 at location 3 contributes to the parity $2^0$ parity bit and to the $2^1$ parity bit. The 1 at location $7= 2^2+ 2^1+2^0$ contributes to the parity the $2^0$, $2^1$, and $2^2$ parity bits.
- When a code arrives, the receiver checks which (if any) check bits are in error, writing a 1 if the bit is in error and a 0 if it is not. For example, if we receive 00110010001. In addition, to the 3 and 7 locations being on we now have that location 11 is on. As $11 = 2^3+ 2^1+2^0$ it contributes $2^0$, $2^1$, and $2^3$ parity bits making the values in each of these parity locations wrong. The $2^2$ bit is still okay. So we would write down the number (1011) in binary. But 1011 in binary is 11. So by writing in this manner, we have gotten the bit position which was messed up!

# Burst Errors

- Often errors come in bursts rather than being randomly distributed.
- To handle burst error of up to k errors using a Hamming code, we can arrange a sequence of k consecutive codewords into a matrix, one codeword per row.
- We can then transmit the data column by column in k bit blocks.
- The receiver then receives data until it has gotten kn bits of data.
- As long as no more than k bits of error occurred in one burst, one only has at most one column in each row damaged.
- So than we can reconstruct this damaged column as before.



# Example Hamming Code to Encode 7 bit ASCII as an 11 bit Code



# Error Detecting versus Error Correction

- For wireless it makes sense to use error correction, but for copper or fiber it typically does not.
- As an example if the error rate is $10^{-6}$ per bit.
- If one transmits 1000 bit block, then using a hamming code takes 10 check bits. So sending 1Mbit of data involves sending 10,000 check bits.
- To detect a single bit error, one can have 1 parity bit/block. So to send 1Mbit of data only need to send 2001 extra bits. (Assuming one error occurred and we had to resend that block.)

# Vertical Parity Checks

- If a block is badly garbled, there is a 50% chance we won't detect the error because there were an even number of errors.
- To handle this, can arrange blocks into k rows of length n and then add a row at the end with parities of each column in addition to the parities on each row.
- Data is still sent row-wise. But now we will be able to detect burst of errors up to length n.