ECE 333: Introduction to Communication Networks
Fall 2002

Lecture 6: Data Link Layer II

- **Error Correction/Detection**

Notes

In Lectures 3 and 4, we studied various impairments that can occur at the physical layer. These impairments can result in occasional errors in the received bit stream, e.g. a 0 being sent and a 1 being received. We emphasize that no matter how well engineered the physical layer is, there will always be some chance for bit errors to occur.

A common approach for modeling bit errors is in a probabilistic framework. For example, assume that each received bit is in error independently with some probability $p << 0.5$. We will refer to this as a ***single bit error model***. In many cases, bit errors tend to occur in ***bursts***, in which case it would be better to model bit errors as being dependent, i.e. given that a bit is in error the probability that the next bit is also in error will increase.

In networks, it is often desirable to devise methods to control the errors that occur in the arriving data. We will discuss this issue in the next few lectures. Though we are discussing this at the DLL, error control methods can be employed at various layers within a network.

Bit Errors

- Bit errors occur at physical layer.
- Sources of errors:
  - Physical layer impairments – noise, fading, cross-talk
  - Synchronization errors (loss of framing)
- Two classes of errors:

Single bit errors:	common inside computers (memory), on optical links.
Burst errors:	common in communications systems or in disk transfers on computers.

Error Control

What to do when errors occur?

Some options:

- Try to ***correct*** errors at the receiver.
- Try to ***detect*** errors at the receiver and either
  - Signal to sender so the frame can be ***retransmitted***
  - Signal up to higher layers at the receiver so that higher layers can compensate for the errors (e.g. music CD: play a few microseconds of silence)

Notes:

To either detect or correct errors at the receiver requires adding ***redundancy*** (extra bits) to each packet accepted from the network layer. For example, suppose that a packet contains m bits and each of the 2^m m bit sequences is a possible packet. Without adding any redundancy, it would be impossible to tell if an error occurred or not. Adding redundancy in a way such that errors can be detected and/or corrected is the subject of ***coding theory***. We will look at some simple examples of this approach in the following.

Still assuming that each packet contains m bits, suppose we encode that packet into a sequence of n bits where $n>m$. This encoding is often done by adding n-m additional bits to the original packet. For each packet of m bits, the resulting sequence of n bits is called a ***codeword***; the set of all 2^m codewords along with the encoding rule is called a ***block code***. Note there are 2^n possible sequences of n bits, but only 2^m of them are codewords. Thus if the received frame contains a sequence which is not a codeword, we know an error has occurred.

Ex. 1: (Single) Parity check code:

A ***single parity check code*** is formed by adding an additional bit to each packet to form a codeword, so that each codeword has an even number of 1's. For example, if a packet contains m=3 bits, the packet 010 would be encoded into the codeword 0101. This can also be expressed as saying that the sum of the bits (mod 2) in a codeword is equal to 0. (Recall that addition (mod 2) satisfies the following rules: 0+0=1+1=0 and 1+0=0+1=1.) Upon receiving a codeword, the receiver will check if

the mod 2 sum of the bits is equal to zero; if no bit errors occur, this will be true. If a single bit in the codeword is changed then the mod 2 sum of the bits will be equal to 1 and this error will be detected. However, if two bits are in error then this will not be detectable. Indeed, with a single parity check code, any odd number of bit errors can be detected and any even number of bit errors will not be detected.

Consider the single bit error model where the probability of bit errors is p= 0.01. For packets with m= 5 bits, without any coding no errors can be detected. Thus the probability of an undetected error is equal to the probability at least one bit is in error which is equal to 1 - Pr(no bits in error)= $1-(1-p)^5 = 0.049$. However if a single parity check code is used we have

$$\Pr(\text{undedected error}) = \Pr(\text{even number of errors})$$
$$= \binom{6}{2} p^2 (1-p)^4 + \binom{6}{4} p^4 (1-p)^2 + p^6 \approx 0.0014.$$

We have significantly reduced the probability of an undetected error. The cost of this is a reduction in the rate that we can send "useful data". Specifically of every 5 bits of useful data we send, we have to transmit a 6 bit codeword. Thus, we have reduced the data rate by 5/6. Note that this reduction in error probability depended strongly on the assumed independence of bit errors - for example with burst errors - it would be equally likely that an even or odd number of errors occurred, in this case the probability of undetected errors would only be reduced by approximately 1/2.

To further lower the error rate more redundancy is needed.

Ex. 2 Repetition code

A repetition code simply repeats the bits in the packet a given number of times. The following gives an example of a repetition code for a case where each packet contains m=2 bits and these bits are each repeated twice:

packet	codeword
00	000000
01	010101
10	101010
11	111111

In this case if the received codeword is not in the above table then an error will be detected. Note, all one and two bit error patterns can be detected with this code, as well as some, but not all 3 bit errors. Suppose that the sequence 000001 is received and you are told that a single bit error occurred. Then the only possible codeword that could have been sent is 000000. In this manner, the above repetition code can be used to ***correct*** all single bit errors. Again, correcting errors in this way makes sense if it is more likely that a single bit error occurred than two or more bit errors, as in the single bit error model. When a code is used to correct errors in this manner it is sometimes referred to as ***Forward Error Correction(FEC).***

Hamming distance

A useful quantity to discuss the error correcting and detecting properties of a code is the notion of the ***hamming distance*** between two binary sequences of a given length. This is defined as the number of positions in which the bits are different. For example the Hamming distance between the sequences 1100 and 0110 is 2. Note that the Hamming distance between a codeword and a corresponding received sequence is equal to the number of errors in the received sequence.

The ***Hamming distance of a code*** is defined to be the minimum of Hamming distances between every pair of valid codewords. The Hamming distance of a code is the smallest number of errors that can go undetected.

What is the Hamming distance of parity check code? Repetition code?

Suppose k bit errors occur in a transmitted codeword. Thus the transmitted codeword has hamming distance of k from the received sequence. If every other codeword has hamming distance greater than k from the received sequence, then the k bit errors can be corrected.

Thus $\left\lceil \frac{\text{hamming dist.}}{2} \right\rceil$ is the smallest number of errors that cannot be corrected with a given code.

Note for any code you can always detect more errors than can be corrected.

Single Bit Error Correction

Suppose we have m bits in our message and want to be able to correct all single bit errors using codewords of length n. Next, we consider how large n must be to accomplish this.

Notice that each codeword has n bit patterns at Hamming distance 1 from it. To correct single bit errors, none of these patterns can be another codeword or at distance 1 from another codeword.

Hence, each of the 2^m messages requires $n + 1$ bit patterns dedicated to it (the n bit patterns at distance 1 plus the valid codeword itself.). Since there are 2^n possible sequences of length n, it must be that

$$(n + 1)\ 2^m \leq 2^n$$

Letting $r = n - m$, this can be rewritten as $m \leq 2^r - r - 1$.

which gives a lower bound on r (and hence n) required.

e.g. $m = 7 \Rightarrow r \geq 4$, $m = 65536 \Rightarrow r \geq 17$

Hamming Code

Hamming codes are single error correcting code that meets the above bound in a clever way. We give one construction of such a code.

Construction:

Number the bit positions of each codeword starting with 1.

Assume that the bit positions that are:

- Powers of 2 correspond to check bits (redundancy)
- Non-powers of 2 correspond to the information bits

The check bits are parity checks on subsets of the data, where the data bit in the k^{th} position is included in the parity checks corresponding to its binary expansion.

E.g. the 5th bit will be included in the 1st and 4th parity check.

Ex: For 7 information bits (e.g. ASCII characters) we need 4 check bits.

	1	2	3	4	5	6	7	8	9	10	11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X

Check bit in position 1 is even parity on 1, 3, 5, 7, 9, 11

ASCII a = 1 1 0 0 0 0 1 => _ _ 1 _ 1 0 0 _ 0 0 1

Since each number has a unique binary expansion, no two positions are involved in same checks.

Thus, for any single error, the check bits that are wrong uniquely indicate the bit in error.

Note: Minimum distance is 3, so any double error is detectable, but not always correctable

Error Detection vs. Correction

Error correction is usually done at the physical layer; while at the DLL and higher layers, the common approach is detection and retransmission.

One reason for this is that the performance of error correcting codes is highly dependent on characteristics of the physical channel and thus better suited for the physical layer.

(To approach the Shannon limit, which was discussed in lecture 3, requires the use of powerful error correcting codes at the physical layer.)

Another reason is that at the DLL, errors are often rare and occur in bursts. It requires less overhead to detect a long burst than too correct one. However when an error is detected, then the frame must be retransmitted, which is also an overhead, but this only occurs when errors are detected instead of in every packet.

The choice between error detection and correction will depend on the setting. For example if there is no reverse communication channel available then retransmissions are not an option. Also for links with long delays, the delay required for retransmission may not be acceptable. These two approaches are not mutually exclusive; and in many cases both are performed. For example, in a network it is common for both error correction to be done at the physical layer and error detection/retransmission to be implemented at the DLL.