EECS 121: Coding for Digital Communication and Beyond **Fall 2013**

Lecture 6 — September 26

Lecturer: Anant Sahai *Scribe: Diivanand Ramalingam*

Announcement: The first midterm will be on Thursday, October 3, in-class.

6.1 Last Lecture Recap: Previously on Digital Comm...

Recall that in the previous lecture we discussed an Encoding/Decoding mechanism known as Orthogonal Signaling. Suppose we have k bits, then we must be able to transmit 2^k symbols or messages. We then associate each message with a waveform $X_i(t)$ of duration [0,T], T is some positive constant, so if we have messages $m_1, ..., m_{2^k}$, then $m_i = X_i(t)$ for $1 \leq i \leq 2^k$. Furthermore these waveforms were orthogonal to each other, meaning that the inner product between any two of these waveforms is 0. Why is this useful?

It provided us with a very simple decoding mechanism, take the inner product of the received signal $Y = X_m + N$, X_m is the message we sent, where N is noise which we model as Additive White Gaussian Noise, with each of the X_i's. Clearly if there were no noise all the inner products would be 0 (Do you see why?) except for the correlator representing the waveform that was actually sent since then you are doing an inner product of a signal with itself, which returns the square of the magnitude of the signal and thus you will know what was the original signal that was sent . But with noise, Y is not necessarily the waveform X that was sent. Assuming the noise is much 'weaker' than the waveform (aka high signal to noise ratio), the inner product will still be very close to 0 for the correlators that represent the waveforms that weren't sent and the inner product with the correlator that represents the waveform that was sent may have a slightly smaller value than before but will still be significantly high and thus this disparity allows one to create a threshold to make a decision rule on what X_i we received given Y. Furthermore we showed that the probability of error would keep decreasing as we sent more bits together.

For a more detailed explanation on this I recommend you read the lecture notes right before this one. Now you may ask "So if orthogonal signaling works, why are we changing it? Don't fix what isn't broken right?" Well unfortunately orthogonal signals as you may have noticed use way too much bandwidth to be practical in production since in today's hot frequency spectrum real estate market, we just cannot afford to have such a large vector space.

Basic Communication Channel Model:

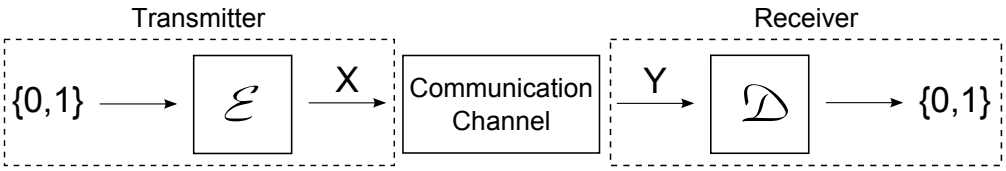

Figure 6.1.

Encoding:

Messages	Waveform on [0,T]	
1	$X_1(t)$	
2	$X_2(t)$	All Waveforms are Orthogonal to Each Other
.		
.		
.	.	
.		
.		
2^k	$X_{2^k}(t)$	

Decoding ($N = 2^k$):

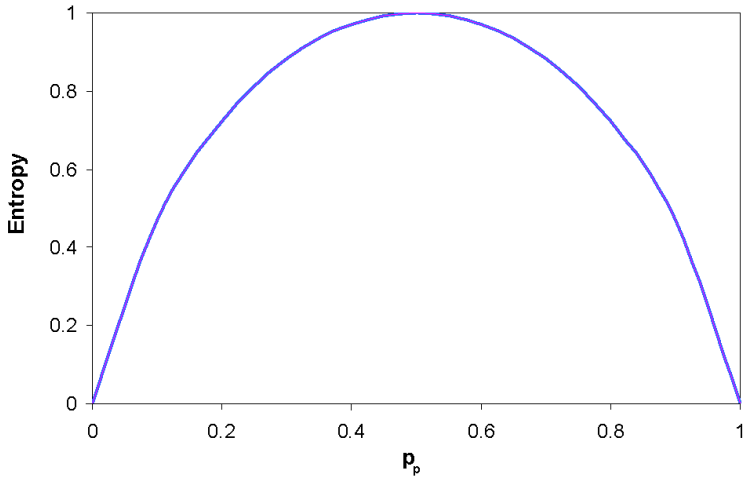

Figure 6.2.

6.2 Beggers Can't Be Choosers: Settling for Independence

So orthogonality is too expensive as realized in the previous section due to its large bandwidth requirements, but played a pretty big role during decoding in the previous encoding/decoding scheme. Thus we ask ourselves "how can I get orthogonality-like behaviour and benefits without orthogonality?" Well your first thought might be "Is there something else that behaves like orthogonality?". Basically we want some form of "approximate orthogonality", such that the inner product between two signals is basically zero if they are not equal, and our analysis of the distance between the noise and signal being far enough apart relative to their spread during error analysis will still basically hold. By removing the restriction of using orthogonal signals we thus can use less bandwidth to transmit our messages.

Now those with some exposure to a class on introductory probability theory or statistics might guess that "independence" (defined in the probability theory sense) is orthogonality's cheaper substitute. This would be based on the idea that if a variable is just a positive scalar multiple of another variable, then in a linear algebra sense, one vector is just a scalar multiple of the other so the 'angle' between them is 0. Now as that angle increases and approaches 90 degrees (right angle, orthogonal), the component of one vector that is a scalar multiple of the other (the projection of that one vector onto the other) becomes smaller and smaller until you have a right angle (orthogonal), where that component that is in the other vectors direction is 0. Also note you have already seen a relationship between orthogonality and independence in Gaussian type sampling since orthogonality resulted in the independence of Gaussian random variables. Thus we will now attempt to construct an encoding/decoding scheme based on independence as opposed to orthogonality and hope this resolves our bandwidth issue while still having the same property of being able to make the probability of error as small as we want by sending more bits together, increase the SNR (sending higher energy signals, aka bigger amplitudes), etc.

6.3 There are two kinds of people: those who analyze bits and those who don't

Although we are sending messages/symbols, lets not forget that the error in these messages comes from the fact that at least one of the components that make up this message (a bit) has been flipped. A bit, or binary digit, can only take on the values 0 and 1, thus all noise can do to "mess" that bit up is flip it. Thus instead of thinking of our X's and Y's as continuous waveforms, let's think of them as discrete signals made up of bits, and lets model noise as being a Bernoulli random variable with probability of being 1 as p (and thus probably of being 0 as (1-p)). Lets use use the notation $X_i[j]$, where the subscript $1 \leq i \leq 2^k$ represents what encoded message is being sent and $1 \leq j \leq S$, where S is the number of bits in the

encoded message, represents what bit (counting from the right) in that encoded message we are looking at, so $X_i[j]$ is the j^{th} bit of the i^{th} message. Too many words can get confusing so here's a table below:

Encoding:

Messages	Waveform on [0,T]	
1	$\underline{X_1[1]}$ $\underline{X_1[2]}$ ... $\underline{X_1[S]}$	
2	$\underline{X_2[1]}$ $\underline{X_2[2]}$ ... $\underline{X_2[S]}$	So a k-bit message is now encoded as an S-bit signal.
.		
.		
.	.	
.		
.		
2^k	$\underline{X_{2^k}[1]}$ $\underline{X_{2^k}[2]}$ ... $\underline{X_{2^k}[S]}$	

Your next question is probably "What should S be? My mind is going have to a hard time if we have too many free variable lying around." I could not agree more thus lets first think of values for S that our intuition first tells us probably will not work.

Case: S < k
This means the signal I'm sending through the channel has fewer bits than the original message itself so clearly I cannot recover the original message no matter what (do you see why?)

Case: S = k
This means my encoded message is the actual message itself, aka no encoding or form of redundancy. In a world without noise this should be fine right? Unfortunately growing up with two younger brothers who loved to ask their big brother to play with them even when he had homework due the next day I can tell you from experience the world is full of noise and thus if a bit gets corrupted we have less uncorrupted bits than the number of bits in the original message and thus determining the original signal should be impossible just like in the previous case.

This leaves us with only one case/option left: S > k, which makes sense since we want to send more bits than necessary so that if some of them get flipped we still should hopefully have at least k unflipped bits and it feels like we could maybe determine the original signal with high confidence using that. So what should we make S? For the sake of Analysis lets make $S = 2k$, thus returning me back to one free variable which puts my simple mind who doesn't like to remember things at ease.

So do we have an encoding strategy now? Well almost but we still haven't decided how to determine what those code bits in the encoded message should be. We'll get back to

that but for now lets briefly discuss how the decoding should work. In the old setting we considered an inner product strategy where we took the inner product of the received signal Y with each X_i (bank of correlators) and had a threshold the inner product that exceeded the threshold corresponds to the message we say was sent.

Well, if you aren't me, then you probably have an avid social/dating life and really need to get to dinner with your significant other, so you might say "Well Diiv, if the worse that the noise can do is flip bits since we live in this magical binary modulo 2 world, why can't I just count the number of bit differences (or symmetrically the number of bit similarities) and use that as my "inner product-like" metric?" Good answer reader and I agree, so if I understand you correctly you say this should be our decoding strategy:

Decoding:

Let $diff(x, y)$ be a function that takes in two bits that returns 1 if those bits are different, and 0 other wise (Quick mind probe: do you know of a bit operation that does this, what is it called?), and $same(x, y)$ also take in two bits and returns 1 if those bits are the same, and 0 otherwise. Given a codeword $X_m[i]$ to be used as a correlator and our received signal $Y[i]$, we compute:

$\sum_{i=1}^{2^k} diff(X_m[i], Y[i])$ (Higher sum is worse) OR equivalently,
$\sum_{i=1}^{2^k} same(X_m[i], Y[i])$ (Higher sum is better)

Yay we now have our "inner product"! Do you see what our decision rule will be for this new inner product? If you don't see it right away, play some jeopardy music and think for while or maybe ask a friend. So this takes care of decoding! (Well technically we still have to threshold but we will get back to that later)

6.4 When in Doubt, Flip a Fair Coin: The Power of Bernoulli $\frac{1}{2}$

I hope you didn't forget that we still haven't decided on a way to encode our messages. What should the value of the j^{th} bit of the i^{th} message be? Well I know it can only be 0 or 1, but that is still two choices. If I pick 1 why is it better than 0, if I pick 0, why is it better than 1? I mean nobody likes to be a 0 and everyone wants to be number 1, but this is the binary world, not the real world, and it couldn't care less. You know what I do when I cannot decide between two things? I flip a fair coin. So I'm going to say the bit will be 0 if I get tails, and 1 if I get heads. That's right. I am going to decide each and every bit in the code book with fair independent coin tosses. Therefore the value of the j^{th} bit of the i^{th}

message is determined by a Bernoulli random variable with probably $p = \frac{1}{2}$

"But Diiv, what if two of the code strings that correspond to different message end up being the same because the probability of that happening is not zero so it could happen!" says you. Well my friend, first of all I ask you what is the probability that 2 $2k$ bit strings are exactly the same, assuming each bit was chosen with fair coin tosses? If $k = 100$, I'd have a much better chance of winning the California lottery. I could give you the answer but I'll leave it as a quick mind probe exercise.

Before we start getting into the mathy stuff and my tone in this paper becomes a little more formal, lets warmup by deciding what kind of random variable $Y[i] = X_m[i] + N[i]$ will be.

So recall that we assumed that our noise bit was $Bernoulli(p)$ and our $X_m[i]$'s are i.i.d $Bernoulli(\frac{1}{2})$, and that are noise is independent of the message sent. I leave it as an exercise to the reader to see that the sum of two independent Bernoulli variables where one of them is $Bernoulli(\frac{1}{2})$ and the other is $Bernoulli(p)$ is $Bernoulli(\frac{1}{2})$....Fine maybe I'll give you this one.....

For a particular i

$P(X_m[i] + N[i] = 1) = \text{P}(\ \{X_m[i] = 0 \text{ and } N[i] = 1\} \text{ or } \{X_m[i] = 1 \text{ and } N[i] = 0\}\ )$

$= P(X_m[i] = 0)P(N[i] = 1) + P(X_m[i] = 1)P(N[i] = 0) = \frac{1}{2}p + \frac{1}{2}(1 - p)$

$= \frac{1}{2}(1) = \frac{1}{2}$

Thus $P(X_m[i] + N[i] = 0) = 1 - P(X_m[i] + N[i] = 1) = \frac{1}{2}$

So $Y[i]$ is $Bernoulli(\frac{1}{2})$ for particular i.

6.5 Error Analysis Time: Why? Because incorrectly decoding a bit in front of everyone is sooo embarassing...

Lets get back to thinking about the difference sum we talked about earlier. What kind of random variable is $diff(X_m[i], Y[i])$? (Assume for now on that if I have an index that I am referring only to the value at the index and when I mention the variable without the index it refers to the whole variable). Well lets calculate it:

$\mathrm{P}(diff(X_m[i], Y[i]) = 1) = \mathrm{P}(\{X_m[i] = 0 \text{ and } Y[i] = 1\} \text{ or } \{X_m[i] = 1 \text{ and } Y[i] = 0\})$

$= \mathrm{P}(X_m[i] = 0 \text{ and } Y[i] = 1) + \mathrm{P}(X_m[i] = 1 \text{ and } Y[i] = 0)$

$= \mathrm{P}(X_m[i] = 0)\mathrm{P}(Y[i] = 1 \mid X_m[i] = 0) + \mathrm{P}(X_m[i] = 1)\mathrm{P}(Y[i] = 0 \mid X_m[i] = 1)$

$= \mathrm{P}(X_m[i] = 0)\mathrm{P}(\text{noise flipped bit}) + \mathrm{P}(X_m[i] = 1)\mathrm{P}(\text{noise flipped bit})$

$= \frac{1}{2}(2p) = p$

So $diff(X_m[i], Y[i])$ is $Bernoulli(p)$ and note it is independent of others in the sum (Can you see why?).

That calculation will apply only if we are correlating with the X actually sent, aka X_m. but what about for the X's we didn't send. What is $diff(X_m[i], Y[i])$ in that case? I leave it as an exercise to the reader to confirm that $diff(X_m[i], Y[i])$ is $Bernoulli(\frac{1}{2})$ in that case and also in this case each $diff(X_m[i], Y[i])$ is independent of others in the sum.

Before we move on, note that our difference sum is the counting the number of differences between the two bit strings, or equivalently, counting the number of ones we get as we apply the diff function to each pair of bits. Thus the difference sum for the true message is a binomial random variable with parameters p (probablity that bit could be flipped) and we have $2k$ such trials as samples, so it is $Binomial(p, 2k)$. And for a false X we have the difference sum to be $Binomial(\frac{1}{2}, 2k)$ (Do you see why?)

So I'm not sure if you the reader was a very strong student, an average student, or something in between while in school, and I personally have played all three roles over the course of my 21 years of living (home economics class just was not my thing, you try sewing up a stuffed animal it is pretty hard), but when you took your probability class, regardless of how much attention you paid in it, hopefully the phrase "sum of independent identically distributed random variables" triggers some sort of excitement. If it doesn't, well.....come on dude, why are you going to class if you are not going to pay attention? But I digress. It turns out that the sum of independent identically distributed random variables is a random variable whose pdf converges to the pdf of a normal distribution (I know...pretty crazy right?) whose mean is the expectation of that random variable and whose variance is the variance of that random variable as the number of elements in this sum approaches infinity.

That said, we now conclude that for the true message, let D = difference sum, D is distributed as a Gaussian with mean $2kp$ and variance $2kp(1-p)$ [this is the mean and variance of a binary with parameters p and $2k$, you should work this out yourself]. For a false message, D is distributed as a Gaussian with mean $\frac{1}{2}$ and variance $\frac{k}{2}$.

To calculate the errors, lets get an idea of these two distributions and how their mean and variance grow and change as k increases, so lets looks at these two variables by scaling D. We do this by defining a new random variable $B = \frac{D}{\sqrt{2k}}$.

Then for a true X, B is Gaussian($\sqrt{2k}p$, $p(1-p)$), and for false X, B is Gaussian($\frac{1}{2\sqrt{2k}}$, $\frac{1}{4}$). The key here is to notice that as k increases, the two B's get further apart from each other which means the noise becomes less likely to affect things, which seems to resemble the "solidarity of bits" advantage that we saw in the orthogonal scheme. Another way to thinking about this is going back to our old random variable D and noticing that the standard deviation increases on the order of $\sqrt{k}$ while the distance between the two D's are increasing on the order of k. So lets have our thresholding rule be based on the standard deviation of D for true X. Thus we can think of our message as a dot on a grid in the center of a square barrier or box around it. Let's call it our decoding box. The size of this box will be on the order of the standard deviation, or the square root of the variance. Think of our received message Y as a dart throw as this grid and the square barrier it falls in will be the X message we decide that we received.

So you must be getting impatient by now and your significant other has probably left the restaurant thinking you stood him or her up, so lets wrap up and actually do the error analysis. We have two error scenarios:

E_1: Noise pushed our Y outside of true X_m's decoding box.
E_2: At least one of the false X_m's moved into our decoding box. The nerve!

So $P(error) \leq P(E_1) + P(E_2)$ by the union bound and thus if we can show that we can make $P(E_1)$ and $P(E_2)$ as arbitrarily small as we want by increasing k, then we will be done. As we make k large, we increase the time to send a message since rate $= \frac{number of input bits}{number of output bits} = \frac{1}{2}$, but our bandwidth is now constrained because by giving up orthogonality in favor of independence, we can send approximately orthogonal signals (aka signals that are much "closer together" in the spectrum than orthogonal signals so bandwidth is more constrained). So this resolves the bandwidth issue we had with orthogonal signaling, although the size of our codebook increases exponentially with k which is a problem we resolve in next lecture's encoding/decoding mechanism.

Since the standard deviation of both D's is growing on the order of $\sqrt{k}$ but the distance between them is growing on the order of k, this implies that as $k -> \infty$, $P(E_1) -> 0$

Now what about E_2? Well we can again use a union bound to see that:
P(E_2) $\leq 2^k$ P(some false X went into our decoding box) $= 2^k \frac{||\square||}{2^{2k}}$, where $\square$ is our decoding

box, so the $||\Box||$ is it's size. Why is it simply 2^k times the size of the box divided by the size of the entire space? Note that our Y is uniformly distributed, in the sense that it can be any possible $2k - bit$ string with equal probability (How many such strings are there? 2^{2k}) and there are 2^k X's.

How big is this box? First notice that the size of the box should be related to the number of $2k - bit$ vectors who have $2kp$ ones. To answer that we recall the AEP form of the Weak Law of Large Numbers for $Bernoulli(p)$ random variables:

Let $A_\epsilon^k = \{\mathbf{X} \in \{0,1\}^k | (p-\epsilon)k \leq X_i \leq (p+\epsilon)k$. Then AEP states:

$\lim_{k\to\infty} P(\mathbf{x} \in A_\epsilon^k) = 1$, where $\mathbf{x} \in \{0,1\}^k$ and

$|A_\epsilon^k| \leq 2^{k(H(p)+\epsilon')}$, where $H(p) = plog_2(\frac{1}{p}) + (1-p)log_2(\frac{1}{(1-p)})$ is called the entropy function and $\epsilon' > 0$. A graph is of $H(p)$ for various values of p for a Bernoulli random variable is shown below:

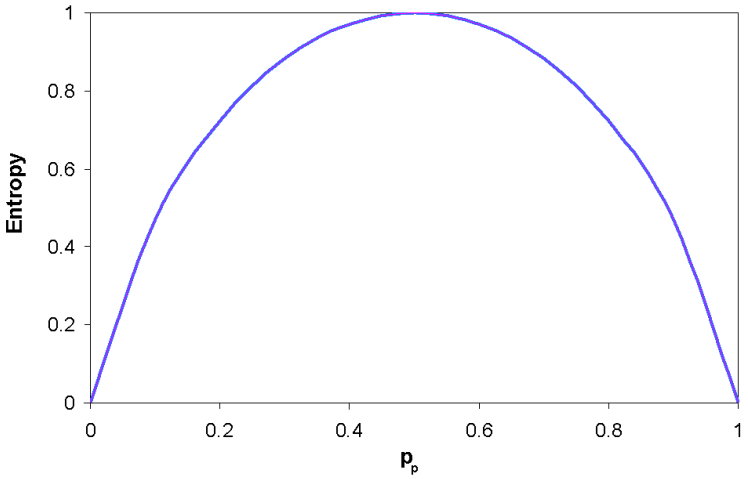

Figure 6.3.

Thus $||\Box|| = |A_\epsilon^{2k}|$, and so if $p < \frac{1}{2}$, $P(E_2) -> 0$ as $k -> \infty$. (Make sure you see why).

Alright you've read enough, so get out and go breath some fresh air.