PROFESSOR: As this says, and all the handouts say that this is 6.450. It's a first year graduate course in the principles of digital communication. It's sort of the major first course that you take as a graduate student in the communication area. The information theory course uses this as a prerequisite, uses it in a rather strong way. 6.432, the stochastic process course uses it as a prerequisite. 6.451, which is the companion course which follows after this uses it as a prerequisite. It's sort of material that you have to know if you're going to go into the communication area. It deals with this huge industry, which is the communication industry.

You look at the faculty here in the department and you're a little surprised to see that there are so few faculty members in digital communication, so many faculty members in the computer area. You wonder why this is. It's sort of this historical accident, because somehow or other the department didn't realize as these two fields were growing, the one that looked more glamorous for a long time was the computer field. Nobody in retrospect understands why that is. But at the same time that's how it is. You will not see as much excitement here. You also will see that because of the big bust in the year 2000, there's much less commercial activity in the communications field now. As students, you ought to relish this. You ought to really be happy about this if you want to go into the communication field, because at some point in the future, all of the failure to start new investments, and all of the large communication companies means a very explosive growth is about to start. So I don't think this is a dead field by any means.

Looking back historically, every time the communication field has seemed dead, and I can think of three or four such times in history, those have been the exact times where it was great to get into the field. There was a time in the early '70s when all the theoreticians who worked in communication theory all were going around with

long faces saying everything that we can do has been done, the field is dead, nothing more to do, might as well give up and go into another field. This was really exactly the time that a number of small communication companies were really starting to grow very fast because they were using some of the modern ideas of communication theory and they were using these modern ideas to do new things. You now see companies like Qualcomm that were started then, Motorola of whom major divisions were started back then. An enormous amount of activity was starting just then, partly because the field seemed to be moribund and the theoreticians were turning and deciding well, I guess we have to do something practical now. If you want to find a good practical engineer, if you wind up in industry, if you wind up as an entrepreneur or if you wind up climbing the ladder and being a chief executive, if you want to find somebody who will really do important things, find somebody who understands theory who has decided that they suddenly want to start to do practical things. Because if you point to the people who have really done exciting things in this field, those are the ones who have done most of it.

OK, so anyway, the big field, it's an important field. What we want to do is to study the aspects of communication systems that are unique to communication systems. In other words, we don't want to study hardware here, we don't want to study software here, because hardware is pretty much the same for any problem you want to look at. It's not that highly specialized, and software is not either. So we're not going to study either of those things, we're going to study much more the architectural principles of communication. We're going to study a lot of the theory of communication, and I want to explain why it is that we're going to do that as we move on. Because it's not at all clear at first why we want to study the silly things that we'll be studying in the course, and I want to give you some idea today of why we're pushed into doing that. As we start doing these things, I'll give you more of an idea.

I know when I was a student I couldn't figure out why I was learning the things that I was learning. I just found that some things were much more interesting than others, and the things that tended to be interesting were the more theoretical things, so I

started looking at them harder and thought gee, this is neat stuff. But I had no idea that it was ever going to be important. So I'm going to try to explain to you why this is important, and something today about what the proper inter-relationship is of engineering and theory.

As another example of my own life, for a long time I thought -- this is rather embarrassing to say -- but I was a theoretician and not much of an engineer for a very long time. And I kept worrying about this and saying well the stuff I'm doing is fun and I enjoy doing it, but why should anybody pay me for it? Of course, I was being paid by MIT, I was being paid as a consultant, but I felt I was robbing all these people. And finally, I decided a rationale for all of this. Most people felt that theory was silly, and therefore, they would keep asking me well, if you're so smart why aren't you rich? I didn't have any very good answer to that because I felt I was smart because I understood all this theory, but I wasn't rich. So I thought gee, what I've gotta do is do enough engineering to get rich, and then when people ask me that question I can say I am. I would suggest to all you, if you have theoretical leadings, to focus on some engineering also so you can become rich, not because there's anything to do with the money, but just because it lets you hold your head up high in a society that values money a lot more than values intellect. So it's important.

OK. This is a part of this two-term sequence. One comment here is I'm not sure that whether 6.451, which is the second part of this sequence, is going to be taught in the spring or not. It might not be taught until spring of the following year. If a large number of you decide that you really want to do this and do it now, make yourselves heard a little bit because the idea of a lot of people want this course will have something to do on whether somebody manages to teach it or not.

As I was starting to say, theory has had a very large impact on almost every field we can think of. I mean you think of electromagnetism, you think of all these other fields and theory is very important in them. But in communication systems it is particularly important. It's partly important because back in 1948, Claude Shannon who was the inventor of information theory -- you usually don't invent theories, but he really did.

This came whole cloth out of this one mind. He had these ideas turning around in his head for about eight years while the second World War was going on. He was working on other things, which were, in fact, very important things, and he was doing those things for the government. He was working on aircraft control and things like that. He was working on cryptography. But he was just fascinated by these communication problems and he couldn't get his mind off them. He took up cryptography because it happened to use all the ideas that he was really interested in. He wrote something about cryptography during the middle of the war, which made many people believe that he had done his cryptography work before he did his communication work. But in fact, it was the other way around. This was purely the fact that he could write about cryptography, and he never liked to write, so he didn't want to write about communication because it was a more complicated field and he didn't understand the whole thing.

So for 25 years it was a theory floating around. You will learn a good deal of what this theory is. Most graduate courses on communication don't spend much time on information theory. This was not a mistake many, many years ago, but it is a mistake now, because communication theory at its core is information theory. Most people realize this. Most kids recognize it, most people in the street recognize it. When they start to use a system to communicate with the internet, what do they talk about? Do they talk about the bandwidth? No. They might call it bandwidth but what they're talking about is the number of bits per second they can communicate. This is a distinctly information theoretic idea, because communication used to be involved with studying different kinds of wave forms. Suddenly, at this point, people are studying all these other things and are based on Claude Shannon's ideas, therefore, we will teach some of those ideas as we go.

As a matter of fact, I was talking a little bit about the fact that there's been a sort of a love-hate relationship by communication engineers for information theory. Back in 1960, a long, long time ago when I got my PhD degree, the people at MIT, the people who started this field, many of them told me that information theory was dead at that point. There'd been an enormous amount of research done on it in the ten years before 1960. They told me I should go into vacuum tubes, which was the

coming field at that time. Two points to that story. If people tell you that information theory isn't important, please remember that story or else tell them to start to study vacuum tubes. The other point of it -- well, I forgot what the other point of it is so let's go on.

Anyway, information theory is very much based on a bunch of very abstract ideas. I will start to say what that means as we go on, but basically it means that it's based on very simple-minded models. It's based on ignoring all the complexities of the communication systems and starting to play with toy models and trying to understand those. That's part of what we have to understand to understand why we're doing what we're doing.

A very complex relationship between modeling, theory, exercises and engineering design. Maybe I shouldn't be talking about that now and maybe the notes shouldn't talk about it, but I want to open the subject because I want you to be thinking about that throughout the term. Because it's something that most people really don't understand. Something that most teachers don't understand. It's something that most students don't understand. When you're a student, you do the exercises you do because you know you have to do them. There was a sort of apocryphal story awhile ago that somebody told me about a graduate student, not at MIT, thank God, who went into see the person teaching a course on wireless and saying, I don't want to know all that theory, I don't want to have to think about these things, just tell me how to do the problems.

Now, we will explain the enormous stupidity of that as we move on. Part of the stupidity is that the problems that you work on as a graduate student, even in your thesis, the problems that you work on are not real problems. The problems that we work on everywhere in engineering are toy problems. We work on them because we want to understand some aspect of the real problem. We can never understand the whole problem, so we study one aspect, then we study another aspect. The reason for all these equations that we have, it's not a way to understand reality. It really isn't. It's a way to understand little pieces of reality. You take all of those and after you have all of these in your mind, you then start to study the real engineering

problem and you say I put together this and this and this and this and this -- this is important, this is not so important, here, this is more important. And then finally, if you're a really good engineer, instead of writing down an equation, you take an envelope, as the story goes, and you scribble down a few numbers, and you say the way this system ought to be built is the following.

That's the way all important systems get built, I claim. Important systems are not designed by the kinds of things you do in homework exercises. The things you do in homework exercises are designed to help you understand small pieces of problems. You have to understand those small pieces of problems in order to understand the whole engineering problems, but you don't do simulations to build a communications system. You understand what the whole system is, you can see it in your mind after you think about it long enough, and that's where good system design comes from. Simulations are a big part of all of this, writing down equations are a big part of it. But when you're all done, if you design an engineering system and you don't see in your mind why the whole thing works, you will wind up with something like Microsoft Word. That's the truth.

So the exercises that we're going to be doing are really aimed at understanding these simple models. The point of the exercises is not to get the right answer. The answer is totally irrelevant to everything. What's important is you start to understand what assumptions in the problem lead to what conclusions. How if you change something it changes something else. That's when you're using system design, because in system design you can never understand the whole thing. You have to look at what parts of it are important, what parts of it aren't important, and the only way you can do that is having a catalog of the simple-minded things in the back of your mind. Practical engineers get that by dealing with real systems day-to-day all their lives. They see catastrophes occur. Those things tell them what to avoid and what not to avoid. Students can do this in a much more efficient way. Obviously, in the practical experience also, but the experience that you get from looking at these exercises and looking at this analytical material in the right way, mainly looking at it is how do you analyze these simple toy models, how do you make conclusions from

them is what let's you start being a good engineer.

I'm stressing this because at some point this term, all of you, all of you at different times are going to start saying why the hell am I doing this? Why am I dealing with this stupid little problem? Stupid little problems can get incredibly frustrating at times, because you see something that's so simple and you can't understand it. Then you say well, this simple thing can't be important anyway. What I really want to do is understand what this overall system is, and you give up at that point. Don't do that. Sometimes there's a very difficult question about what you mean by something being simple. I will sometimes say that something is very simple, and I will offend many you to whom it's not simple. The point is something becomes simple after you understand it. Nothing is simple before you understand it. So there's that point at which the light goes off in your head at which it becomes simple. When I say that something is simple, what I mean is that if you think about it long enough a light will go off in your head and it will become simple. It's not simple to start with. Other things are just messy. You've all heard of mathematical problems which are just ugly. There are these things of extraordinary complexity. There are things where there just isn't any structure. You see a lot of these things in computer science. I talked about Microsoft Word. Part of the reason it's such a lousy language is because the problem is incredibly difficult. It's incredibly unstructured. So people come up with things that don't make any sense, because that inherently is not simple. OK, these other things that we'll be studying here are inherently simple and the only problem is how do you get to the point of seeing these simple ideas.

OK, so that's enough philosophy. No, not quite enough, a little more of that. All the everyday communication systems that we deal with are incredibly complex in terms of the amount of hardware, the amount of software in them. If you look at the whole system and you try to understand it as a whole, you don't have prayer. There's no way to do it. These things work and they can be understood because they're very highly structured. They have simple architectural principles. What do I mean by architectural principles? It's a word that engineers use more and more. It started off with computer systems because it became essential there to start thinking in terms of architecture. It's exactly the same thing that you mean when you're talking about

a house or a building. Mainly the architecture is the overall design. It's the way the different pieces fit together. In engineering, architecture is the same thing. It's the thing that happens when your eyes fuzz over a little bit and you can't see any of the details anymore, and suddenly what you're doing is you're looking at the whole thing as an entity and saying how do the pieces fit together. Well, what we're going to be focusing on here.

One of the major keys to making these things simpler is to have standardized interfaces and to have layering. And we'll talk a little bit about each of these because our entire study of this subject is based on studying the different layers that we'll wind up with. OK, the most important and most critical interface in a communication system is between the source and the channel. Today, that standard interface is almost always a binary data stream. You all know this. When you talk about your modab, how do you talk about it? 48 kilobits per second if you're an old-fashioned kind of guy. 1.2 megabits if you're a medium technology person, or several gigabits if you're one of these persons who likes to gobble up huge amounts of stuff. That's the way you describe channels -- how many bits per second can you send? The way you describe sources is how many bits per second do you need as a way of viewing that source. When you store a picture, how do you talk about it? You don't talk about in terms of the colors or any of these other things, picture is a bunch of bits. That, at the fundamental level, is what we're doing here. When you deal with source coding, you're talking fundamentally about the problem of taking whatever the source is, and it can be any sort of thing at all -- it can be voice, it can be text, it can be emails, the emails with viruses in it, whatever -- and the problem is you want to turn it into a bit stream.

Then this bit stream gets presented to the channel. Channels and the channel encoding equipment don't have any idea of what those bits mean. They don't want to have any idea of what those bits mean. That's what an interface means. It means the problem of interpreting the bits, the problem of going from something which you view as having intelligence down to a sequence of bits is the problem of the source coder. The problem of taking those bits and moving them from one place to another

is the function of the channel designer, the network designer and all of those things. When you talk about information theory, it's a total misnomer from the beginning. Information theory does not deal with information at all, it deals with data. We will understand what that distinction is as we go on. When you talk about a bit stream, what you're talking about is a sequence of data. It doesn't mean anything. It's just that a one is different from a zero, and you don't care how it's different, it just is.

So the channel input is a binary stream. It doesn't have any meaning as far as the channel is concerned. The bit stream does have a meaning as far as the source is concerned. So the problem is the source encoder takes these bits, takes the source, whatever it is, with its meaning and everything else, turns it into a stream of bits -- you would like to turn it into as few bits as possible. Then you take those bits, you transmit them on the channel, the channel designer understands what the physical medium is all about, understands what the network is all about, and doesn't have a clue as to what all those bits mean. So the picture is the following. That's the major layering of all communication systems. Here's the input, which is whatever it happens to be. Here's the source encoder. The source encoder has to know a great deal about what that input is. You have to know the structure of the input. You have a source encoder for voice and you use it to try to encode a video stream it's not going to work, obviously. If you try to use a source encoder for English and try to use it on Chinese, it probably won't work very well either. It won't work for anything other than what it's intended for. So the source encoder has to understand the source. When I say understand the source, what do I mean? It means to understand the probabilistic structure of the source. This is another of the things that Shannon recognized clearly that hadn't been recognized at all before this. You have to somehow understand how to view what's coming out of the source, this picture, say, as one of a possible set of pictures.

If you know ahead of time that a communication system is going to be sending the Gettysburg Address or one of 50 other highly-inspiring texts, what do you do? Do you try to encode these things, all these different texts? Of course not. You just assign number one to the Gettysburg Address, number two to the second thing that you might be interested in. You send a number and then at the output out comes

the Gettysburg Address because you've stored that there. So that's the idea of source coding. What is important is not the complexity of the individual messages, it's the probabilistic structure that tells you what are the different possibilities. That's what happens when you try to turn things into binary digits. You have one sequence of binary digits for each of the possible things that you want to represent.

Then in the channel you have the same sort of thing. You have noise, you have all sorts of other crazy things on the channel. You have a sequence of bits coming in, you have a sequence of bits coming out. The fundamental problem of the channel is very, very simple. You want to spit out the same bits that came in. You can take a certain amount of delay doing that, but that's what you have to do. In order to do that, you have to understand something about the probabilistic structure of the noise. So both ways you have probabilistic structure. It's why you have to understand probability at the level of 6.041, which is the undergraduate course here studying probability. If you don't have that background, please don't take the course because you're going to be crucified. If you think you can learn it on the fly, don't. You can't learn it on the fly. As a matter of fact, if you've taken the course, you still don't understand it, and you will need to scramble a little bit to understand it at a deeper level in order to understand what's going on here. 6.041 is a particularly good undergraduate course. I think it's probably taught here better than it's taught at most places. But you still don't understand it the first time through. It's a tricky, subtle subject you really need to understand enough of it, If you have no exposure to it. If you've taken a course called statistics and probability, which first teaches you statistics and then tucks in a little bit of probability at the end, go learn probability first because you don't have enough of it to take this subject.

The other part of dealing with these channels is that suddenly we have to deal with 4AA analysis and all of these things, if you don't have some kind of subject in signals and systems. Some computer scientists don't learn that kind of material anymore. Again, you can't learn it on the fly because you need a little bit of background for it. Those two prerequisites are really essential. Nothing else is essential. The more mathematics you know the better off you are, but we will

develop what we need as we go.

Now, we talked about this fundamental layer in all communication systems, which is between source coding and channel coding. You first turn the source into bits, and then you turn the bits into something you can transmit on the channel. Source coding breaks down into three pieces itself. It doesn't have to break down into those three pieces, but it usually does. Most source coding you start out with a wave form, such as with voice, or with pictures you start out with what's more complicated than a wave form, some kind of mapping from x-coordinate and y-coordinate and time, and you map all of those things into something. Our problem here is we want to take these analog wave forms or generalizations of. It turns out that all we have to deal with here is the analog wave forms because everything else follows easily from that. So there's one question. How do you turn an analog wave form into a sequence of numbers? This is the common way of doing source coding. Many of you who have been exposed to the sampling theorem, you think that's the way to do it, this is one way to do it. We will talk a great deal about this. You will find out that the sampling theorem really isn't the way to do it. Although again, it's a simple model, it's the way to start dealing with it. Then you learn the various other parts of that as you go along. After you wind up with sequences, sequences of numbers, we worry about how to quantize those numbers, because fundamentally we're trying to get from wave forms into bits. So the three stages in that process are first go to sequences, go from sequences to symbols -- namely, you have a finite number of symbols, which are quantized versions of those analog numbers which can be anything. Then finally, you encode the symbols into bits.

The coding goes in the opposite order. Here's a picture which shows it better than the words do. From the input wave form, you sample it or something more generalized than sampling it. That gives you a sequence of numbers. You quantize those numbers, that gives you a symbol sequence. You have a discrete coder, which turns those symbols into a sequence of bits in some nice way. Then you have this reliable binary channel. Notice what I've done here. This thing is that entire system we were talking about before. This has a channel encoder in it, a channel, a channel decoder and all of that. But what we've been saying in this layering idea is

when you're dealing with source coding you ignore all of that. You say OK, it's Tom's job who was designing the channel encoder to make sure that my bits come out here as the same bits. If the bits are wrong it's his fault. If the bits are right, and this output wave form doesn't look like the input wave form, it's my fault. So part of the layering idea is we just recreate this whole thing here as the same bits and we don't worry about what happens when the bits are different.

The other part of this is that all of this breaks down in the same way. Namely, at this interface between here and here, the idea is the same. Namely, there's an input wave form that comes in, there's a sequence of numbers here. The job of this analog filter, as we call it, and you'll see why we call it that later, is to take numbers here, turn them back into wave forms. These numbers are supposed to resemble these numbers in some way or other. I can deal with this part of the problem totally separately from dealing with the other parts of the problem. Namely, the only problem here is how do I take numbers, turn them into wave forms. There are a bunch of other problems hidden here, like these numbers are not going to be exactly the same as these wave forms because I have quantization involved here. Since the numbers are not exactly the same, we have to deal with questions of approximation. How to approximations in numbers come out in terms of approximations in terms of wave forms? That's where you need things like the sampling theorem and the generalizations of it that we're going to spend a lot of time with.

Then you go down to this point. At this point the quantizer produces a string of symbols. Whatever those symbols happen to be, but those symbols might, in fact, just be integers whereas the things here were real valued numbers, and in quantizing we turn real numbers into intergers by saying -- well, it's the same way you approximate things all the time. You round off the pennies in your checkbook. It's the same idea. So the problem here is the quantizer produces these symbols here. The job of all of this stuff is to recreate those same symbols here. So the symbols now go into something that does the reverse of a quantizer -- we'll call it a table look-up because that's sort of what it is. So we can study this part of the

problem separately, also. You don't have to understand anything about this problem to understand this problem. Finally, we have symbols here going into a discrete encoder. We have an interesting problem which says how do we take these symbols which have some kind of probabilistic structure to them and turn them into binary digits in an efficient way. If the symbols here, for example, are binary symbols and the symbols happen to be 1 with probability 999 out of 1,000, and zero with probability 1 in 1,000, you don't really just want to take these symbols and map them in a straightforward way, 1 into 1 and zero into zero. You want to look at whole sequences of them. You want to do run length coding, for example. You want to count how long it is between these zero's, and then what you do is send those counts and you encode them. So there are all kinds of tricks that you can use here. But the point is, this problem is separate, and this problem is separate from this problem.

Now what we're going to do in this course is start out with this problem, because this is the cleanest and the neatest of the three problems. It's the one you can say the most about. It's the one which has the nicest results about it. In fact, a lot of source coding problems just deal with this one part of it. Whenever you're encoding text, you're starting out with symbols which are the letters of whatever language you're dealing with and perhaps a bunch of other things -- ask a code, you've generated 256 different things, including all the letters, all the capital letters, all of the junk that used to be on typewriters, and some of the junk that's now in word processing languages, and you have a way of mapping those into binary digits. We want to look at better ways of doing that. So this, in fact, covers the whole problem of source coding when you're dealing with text rather than dealing with wave forms. This problem here combined with this problem deals with all of these many problems where what you're interested in is taking a sequence of numbers or a sequence of whatjamacallits and quantitizing them and then coding. Finally, when you get to the input wave forms and output wave forms, you've solved both of these problems and you can then focus on what's important here.

This is a good case study of why the information theoretic approach works and why theory works. Because if you started out with the problem of saying I want to build

something to encode voice, and you try to do all of these together, and you look up all the things you can find about voice encoding on the web and you read all of them, what you will wind up with, I can guarantee you, is you will know an enormous amount of jargon, you will know an enormous number of different systems, which each half work, you will not have the slightest clue as to how to build a better system. So you have to take the structured viewpoint, which is really the only way to do things to find new and better ways of doing things.

Now there are some extra things. I have over-simplified it. I want to over-simplify things in this course. What I will always do is I'll try to cheat you into thinking the problem is simpler than it is, and then I will talk about all of the added little nasties that come in as soon as you try to use any of this stuff. There are always nasties. What I hope you will all do by the end of the term is find those little nasties on your own before I start to talk about them. Namely, when we start to talk about a simple model, be interested in the simple model, study it, find out what it's all about, but at the same time, for Pete's sake, ask yourself the question. What does this have to do with the price of rice in China or with anything else? And ask those questions. Don't let those questions interfere with understanding the simple model, but you've got to always be focusing on how that simple model relates to what you visualize as a communication system problem. It usually will have some relation but not a complete relation.

Here the problems are in this binary interface. The source might produce packets or the source might produce a stream of data. In other words, if what you're dealing with is the kind of situation where you're an amateur photographer, you go around taking all sorts of pictures, and then you encode these pictures -- what do I mean by encoding the pictures? You turn them into binary digits. Then you want to send your pictures to somebody else, but what you're really doing is sending these packets to someone else. You're not sending a stream of data -- you have 10 pictures, you want to send those 10 pictures. At the output of the whole system, you hope you see 10 separate pictures. You hope there's something in there which can recognize that out of the stream of binary digits that come across the channel, there's some

packet structure there.

Well we're not going to talk about that really. Whenever you start dealing with this problem of source encoding, you also need to worry a little bit about the packet structure. What are the protocols for knowing when something new starts, when something new ends. Those kinds of problems are dealt with mostly in a network course here, and you can find out all sorts of things about them there. But there are these two general types of things -- packets and streams of data. In terms of understanding voice encoding, you can study them both together. Why can you study them both together? Because the packets are long and because the packets are long because they include a lot of data, you have a little bit of end effect about how to start it, a little bit of end effect about how to end it, but the main structural problem you'll be dealing with is what to do with the whole thing. It's an added piece that comes at the end to worry about how do you denote the beginning and the end. That's a problem you have with stream data also, because stream data does not start at time minus infinity. Whatever the stream data is coming from, what it's coming from did not at time, t equals infinity. You just think of it that way because you recognize you can postpone the problem of how do you start it and how do you end it, and hopefully you can postpone it until somebody else has taken over the job.

What the channel accepts is either of these binary streams or packets, but then queueing exists. In other words, you have stuff coming into a channel -- sometimes I'm sitting at home and I want to send long files, I want to send this whole textbook I'm writing to somebody. It queues up, it takes a long time to get it out of my computer and into this other person's computer. It would be nice if I could send it at optical speeds. But I don't care much. I don't care much whether it takes a second or a minute or an hour to get to this other person. He won't read it for a week anyway, if he reads it at all, so what difference does it make? But anyway, we have these queueing problems, which are, again, separable. They're dealt with mostly in the network course. What we're mostly interested in is how to reduce bit rate for sources. How do you encode things more efficiently? Data compression is the word usually given to this. How do you compress things into a smaller number of bits

using the statistical structure of it? Then how do you increase the number of bits you can send over a channel? That's the problem with channels. If you have a channel that'll send 4,800 bits per second, which is what telephone lines used to do. If you build a new product that sends 9,600 bits per second, which was a major technological achievement back in the '70s and '80s, suddenly your company becomes a very important communication player. If you then take the same channel and learn how to communicate at megabits over it, that's a major thing also. How did people learn to do that? Mostly by using all the ideas that the people used in going from 4,800 bits per second to 9,600 bits per second. It was the people who did that first job who were the primary people involved in the second job. The point of that is it doesn't really make any difference as an engineer whether you're going from 4,800 to 9,600 or from 9,600 to 19.2 or from 19.9 to 38.4 or whatever it is, or so forth up. Each one of these is just putting in a few new goodies, recognizing a few new things, making the system a little bit better.

Let's talk about the channel now. The channel is given -- in other words, it's not under the control of the designer. This is something we haven't talked about yet. In all of these communication system design problems, one of the things that gets straight is when you're worrying about the engineering of something, what parts of the system can you control and what parts of it can't you control. Even more when you get into layering -- what parts can you control and what parts can other people control. Namely, if I have some source and I'm trying to transmit it over some physical channel and I have two engineers -- one of them is a data compressor, and the other is a channel guy. What the channel guy is trying to do is to find a way of sending more bits over this channel than could be done before. What the data compressor is trying to do is to find a way to encode the source into fewer bits. If the two come together, the whole thing works. If the two don't come together, then you have to fire the engineers, hire some new engineers or do something else or you get fired. So that's the story. The things you can't change here, you can't change the structure of the source usually, and you can't change the structure of the channel.

There's some very interesting new problems coming into existence these days by information theorists who are trying to study biological systems. One of the peculiar things in trying to understand how biological organisms, which have remarkable systems for communicating information from one place to another, how they manage to do it. One of the things you find is that this separation that we use in studying electronic communication does not exist in the body at all. In other words, what we call a source gets blended with the channel. What we call the statistics of the source get very much blended in with what this organism is trying to do. If the organism cannot send such refined data, it figures out ways to get the data that it needs which is essential for its survival or it dies and some other organism takes over. So you find the system which has no constraints in it, and it evolves with all of these things changing at the same time. The channels are changing, the source statistics are changing, and the way the source data is encoded is changing, and the way the data is transmitted is changing.

So it's a whole new generalization of this whole problem of how do you communicate. Here though, the problem we're dealing with is these fixed channels, which you can think of, depending on what your interests are, you can view the canonical channel as being a piece of wire in a telephone network, or even view it as being a cable in a cable TV system, or you can view it as being a wireless channel. We will talk about all of these in the course. The last three weeks of the course is primarily talking about wireless because that's where the problems are most interesting. But in any situation we look at, the channel is fixed. It's given to us and we can't change it. All we can do fiddle around with the channel encoding and the channel decoding. The channels that we're most interested in usually send wave forms. That brings up another interesting question. We call this digital communication, and what I've been telling you is that the sources that we're interested in are most often wave form sources. The channels that we're interested in are most often channels that send wave forms, receive wave forms and add noise, which is wave forms. Why do we call it digital communication? Anybody have a clue as to why we might call all of this digital communication? Yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Because the analog gets represented in binary digits, because we have chosen essentially, because of having studied Shannon at some point, to turn all the analog sources into binary bit streams. Namely, when you talk about digital communication, what you're really doing is saying I have decided there will be a digital interface between source and channel. That's what digital communication is. That's what most communication today is. There's very little communication today that starts out with an analog wave form and finds a way to transmit it without first going into a binary sequence.

So here's a picture of one of the noises that we're going to study. We have an input which is now a wave form. We have noise, which is a wave form. And we have an output, which is a wave form. This input here is going to be created somehow in a process of modulation from this binary stream that's coming into the channel encoder. So somehow or other we're taking a binary stream, turning it into a wave form. Now, think a little bit about what might be essential in that process. If you know something about practical engineering systems for communication, you know that there's an organization in the U.S. called the FCC, and every other country has its companion set of initials, which says what part of the radio spectrum you can use and what part of the radio spectrum you can't use. You suddenly realize that somehow it's going to be important to turn these binary streams into wave forms, which are more or less compressed into some frequency bands. That's one of the problems we're going to have to worry about. But at some level if I take a whole bunch of different binary sequences, one thing I can do, for example, I could take a binary sequence of length 100. How many such binary sequences are there? Has length 100, the first bit can be 1 or zero, second bit can be 1 or zero -- that's four combinations. Third bit can be 1 or zero also -- that makes eight combinations of three bits. There are 2 to the 100th combinations of 100 binary digits. So a theoretician says fine, I'm at those 2 to the 100th different combinations of binary digits into evenly spaced numbers between zero and 1. Then I will take those evenly spaced numbers between zero and 1 and I'll modulate some wave form, whatever wave form I happen to choose, and I will send that wave form. In the absence of

noise, I pick up that wave form and turn it back into my binary sequence again. What's the conclusion from this? The conclusion is if you don't have noise there's no constraint on how much data you can send. I can send as much data as I want to, and there's nothing to stop me from doing it, if I don't have noise. Yeah.

AUDIENCE: So, is that where something that [UNINTELLIGIBLE] basically comes in. Like a difference between one signal and the next signal.

PROFESSOR: Somehow I have to keep these things seperate. I don't necessarily have to separate one binary digit from the next binary digit in time. I could, in fact, do something like creating 2 to the 100th different wave forms or I could do anything in between. I could take each sequence of eight bits, turn them into 1 of 256 wave forms, transmit one wave form, then a little bit later transmit another wave form and so forth. So I can split up the pie in any way I want to, and we'll talk about that a lot as we go on. Now, I split them up into different wave forms, which I send at spaced times. I somehow have to worry about the fact that if I send them over a finite bandwidth, there is no way in hell that you can take finite bandwidth wave forms and send them in finite time. Anything which lasts for a finite amount of time spreads out in bandwidth. Anything which is in a finite amount of bandwidth spreads out in time. That's what you ought to know from 6.003. So we're dealing with an unsoluble problem here.

Well, Nyquist back in 1928 solved that problem. He said well no, you can't make the wave form separate, but you can make the sample separate. People earlier had figured out they could do that with the sampling theorem, but the sampling theorem wasn't very practical, but Nyquist's way of doing it was practical. So, in fact, you can separate these wave forms. But you still have the problem how do you deal with the noise. Well, one of the favorite ways of dealing with noise is to assume that it's something called white Gaussian noise. What is white Gaussian noise? White Gaussian noise is noise which no matter where you look it's sitting there. You can't get away from it. You move around to different frequencies, the noise is still there. You move around to different times, it's still there. It is somehow uniformed throughout time and throughout frequency. Kind of an awkward thing because if I

developed a receiver which didn't have any bandwidth constraint on it, and I looked at this noise coming in, which was spread out over all frequencies, the noise would burn out the receiver. In other words, this white Gaussian noise has infinite power. That shouldn't bother you too much. You deal with unit impulses all the time. A unit impulse has infinite energy also. You probably never thought about it because most undergraduate courses try very hard to conceal that fact from you because they like to use impulses for everything, but impulses have infinite energy associated with them, and that's kind of a nasty thing. But anyway, we will deal with the fact that impulses have infinite energy, and therefore, not use them to transmit data. And we will deal with the fact that this noise has infinite power, and find out how to deal with that. And we will learn how to understand the statistical structure of that noise.

So we'll deal with all of these things when we start studying these channels. As we do that we will find out that no matter what you do on a channel like this, there are some fundamental constraints on how much data can be transmitted. I can take you through a little bit of a thought experiment which will sort of explain that, I think. It's partly what we were just starting to say over here. A simple-minded way to look at this problem is I have binary digits coming in. I take these binary digits and I separate them into shorts sequences of m binary digits each. So, I take the first m binary digits, then I take the next m binary digits, and the next m binary digits and so forth. A sequence of m binary digits, there are 2 to the m combinations of this. So I map these 2 to the m combinations into one of 2 to the m different numbers. Now, you look at this noise and you say I have to keep some separation between these numbers. So, with a separation between the numbers, say the separation is 1, let this define the number 1. That's something we theoreticians do all the time. So I have 2 to the m different levels. I have to send this in a certain bandwidth. The sampling theorem says I can't send things that are more than twice the bandwidth of this system, so I have a bandwidth of 2w. I can send m binary digits in this bandwidth, w. Therefore, I can send bits at a certain rate. This is all very crude but it's going to lead us to an interesting trade-off. I can increase m and by increasing m I can get by with a smaller bandwidth, because I have these symbols, these m bit symbols coming along at a slower rate. So by increasing m, I can decrease the

bandwidth, by increasing m, I can reduce the bandwidth, but by reducing the bandwidth, the power is going up. You can see that as I change m, the power is going up exponentially with m. In other words, the trade-off between bandwidth and power is kind of nasty in this simple-minded picture.

Let me jump ahead. For this added of white Gaussian noise with a bandwidth constraint, Shannon showed that the capacity was w times log of 1 plus the power in the system times the noise power times w. Now, I don't expect you to understand this formula now, and there's a lot of very tricky and somewhat confusing things about it. One thing is that the power and the noise is going up with the bandwidth that you're looking at. Because I said the bandwidth is everywhere, you can't avoid it, and therefore, if you use a wider bandwidth system, you get more noise coming into it. If you have a certain amount of power that you're willing to transmit and a certain amount of noise, and the number of bits per second you can transmit is going up with w. But look at this affect with p here, the effect with p is logarithmic, which says as I increase this parameter m and I put more and more bits into one symbol, this quantity is going up exponentially, which means the logarithm of this is going up just linearly with m.

This gives you, if you look at it carefully, the same trade-off as the simple-minded example I was talking about. That simple-minded example does not explain this formula at all. This formula's rather deep. We probably won't even completely have proven this term. What Shannon's results says is that no matter what you do, no matter how clever you are with this noise that exists everywhere, the fastest you can transmit with the most sophisticated coding schemes you can think of is this rate right here, which is what you would think it might be just from the scaling in terms of w and m, where when you increase m the power goes up exponentially. It's the same kind of answer you get with that simple thought experiment. The fact that you have a 1 in here is a little bit strange, and we'll find out where that comes from. The idea here is that this noise is something you can't beat. The noise is fundamental. It's a fundamental part of every communication system. As I always like to say, noise is like death and taxes, you can't avoid them, they're there, and there's nothing you can do about them. Like death and taxes, you can take care of yourself

and live longer, and with taxes, well, you can become wealthy and support the government and make all sorts of payments and reduce your taxes that way. Or you can hire a good accountant. So you can do things there also. But those things are still always there. So you're always stuck with this. One of the major parts of the course will be to understand what this noise is in a much deeper context than we would otherwise. So, I'll save that and come back to it later.

We have the same kind of layering in channel encoding. When I talk about channeling encoding, I'm not talking about any kind of complicated coding technique. 6.451 talks about all of those. What I'm talking about is simply the question of how do you take a sequence of bits, turn it into a sequence of wave forms in such a way that at the receiver we can take that sequence of wave forms and match them reliabily back into the original bits. There's a standard way of doing this, which is to take the sequence of bits, first send them through a discrete decoder. What the discrete encoder code will do is something like this. And match bits at one rate into bits at a higher rate, and this let's you correct channel errors. A simple example of this is you match zero into zero, zero, zero. 1 into 1, 1, 1. If the rest of your system, namely, if this part of the system makes a single error at some point, this part of the system escapes from it. I sort of put these on one slide but I couldn't. This takes a single zero, turns it into three zeros. These three zeros go into this modulator, which maps this into some wave from responsive to these three zeros. Wave form goes through here, noise gets added, the modulator, the text, those binary digits, comes out with some approximation to these three zeros. Every once in awhile because of the noise, one of these bits comes out wrong, and because they come out wrong this discrete decoder says ah-ha, I know that what was sent/put in was either zero, zero, zero or 1, 1, 1. It's more likely to have one error than to have two errors, and therefore, any time one error occurs I can decode it correctly. Now, that is a miserable code.

A guy by the name of Hamming became very famous for generalizing this just a little bit into another single error correcting code, which came through at a slightly higher rate but still corrected single errors. He became inordinately famous because he

was a tireless self-promoter and people think that he was the person who invented error correction coding. He really wasn't. And his heirs will probably -- I don't know. Anyway, coding of this type, namely mapping binary digits into longer strings of binary digits in such a way that you can correct binary errors, has always been a major part of communication system research. A large number of communication systems work in exactly this way. In fact, the older systems which use coding tended to work this way. This was a very popular way of trying to design system. You would put a total layering across here. Namely, this part of the system didn't know that there was any coding going on here. At this point where you're doing discrete decoding, you don't know anything about what the detector is doing here. It doesn't take a lot of imagination to say if you have a detector here and there's this symbol that comes through here, noise gets added to it, you're trying to decode this symbol, and there's this Gaussian noise, which is just sort of spread around, it's concentrated. Usually when errors occur, errors occur just barely. What you see at this point is you almost flip a coin to decide whether a zero or a 1 was transmitted at this point. If this poor guy had that information, this poor guy could work much, much better. Namely, this guy could correct double errors instead of single errors, because if one bit came through with a clear indication that that's what it was, and the other two bits came through saying well, I can't really tell what these are, then this guy could correct two errors instead of one error.

So this is a good example where this layering in here is really a rotten idea. Most modern systems don't do that strict layering. But a whole lot of modern systems do, in fact, do this binary encoding here and they use the principles that came out of the binary encoding from a long time ago. The only extra thing they do here is they use this extra information from the demodulator. This is something called soft decoding instead of hard decoding. In other words, what comes out of the demodulator is soft decisions, and soft decisions say well, I think it was this but I'm not sure or I am sure and so forth. There's a whole range of graduations. We will study detection theory and we'll understand how that works and we'll understand how you use those soft decisions and all of that stuff and it's kind of neat.

The modulation part of this is what maps bit sequences into wave forms. When we

map bit sequences into wave form, there's this very simple idea. If I take a single bit that's either zero or 1, I map a zero into one wave form, I map a 1 into another wave form, I send either wave form a or wave form b. I somehow want to make those wave forms as different from each other as I can. Now, what do you mean by making wave forms different? I know what it means to make numbers different. I know that zero and 3 are more different than zero and 2 are. Namely, I have a good measure of distance for a sequence of numbers. One of the things that we have to deal with is how do you find an appropriate measure of distance for wave forms. When I find the measure of distance for wave forms, what do I want to make it responsive to? What I'm trying to do is to overcome this noise, and therefore, I have to find the measure of distance which is appropriate for what the noise is doing to me. Well that's why we have to study wave forms as much as we do in this course. What we will find is that you can treat wave forms in the same way as you can treat sequences of numbers. We will find that there's a vector space associated with wave forms, which is exactly the same as a vector space associated with infinite sequences of numbers. It's almost the same as the vector space associated with finite sequences of wave form, and that vector space associated with a finite sequence of numbers is exactly the vector space that you've been looking at all of your lives, and which you use primarily as a notational convenience to talk about a sequence of numbers. What's the distance that you use there? Well, you take the difference between each number in the sequence, you square it, you add it up and you take the square root of it. Namely, you look at the energy difference between these sequences. What we will find remarkably is when we do things right, the appropriate distance to talk about on wave forms, is exactly what comes from the appropriate looking at sequences. In fact, we'll take wave forms and break them into sequences. That's a major part of this modulation process. You think of that in terms of the sampling theorem. Namely, you take a bunch of numbers, you take each number in the sampling theorem and you put a little sin x over x hat around it and you transmit that, and then you add up all of these different samples, which is a sequence of numbers, each of them with these little sin x over x caps around them. The neat thing about the sin x over x caps around them is they all go to zero with

these sample points and it all works. Therefore, the sequences look almost the same as a wave form. We'll find out that that works in general, so we will do most of that.

Modern practice often combines all these layers into what's called coding modulation. In other words, they go further than just this idea of soft decisions, and actually treat the problem in general of what do you do with bits coming into a sequence, into a system, how do you turn those into wave forms that can be dealt with an appropriate way. And we'll talk just a little bit about that and mostly 6.451 talks about that.

I want to talk a little bit right at the end, I'm almost finished, about computational complexity in these communication systems that we're trying to worry about. One of the things that you will become curious about as we move along is that we are sometimes suggesting doing things in ways that look very complex. The word complex is an extraordinarily complex word. None of us know what it means because it means so many different things. I was saying that these telephone systems are inordinately complex. In a sense they aren't because they have an incredible number of pieces, all of which are the same. So you can understand a telephone system in terms of understanding a relatively small number of principles. You can understand these Gaussian channels in terms of understanding a small number of principles. So they're complex if you don't know how to look at them, they're simple if you do know how to look at them. Here what I'm talking about is how many chips do you need to build something? How complicated are those chips? Fundamentally, how much does it cost to build the system? Well, one of the curious things and one of the things that has made information theory so popular in designing communication systems is that we're almost at the point where chips are free. Now, what does that mean? It means if you want to do something more complicated it hardly costs anything. This is sort of Moore's law. Moore's law says every year you can put more and more stuff on a single chip. You can do this cheaply and the chips work faster and faster so you can do more and more complicated things very, very easily.

There are some caveats. Low cost with high complexity requires large volume. In other words, it takes a very long time to design a chip, it takes an enormous amount of lead time -- I mean we're not going to study these details in the course, but I want you to be aware of why it is that in a sense complexity doesn't matter anymore. If you spend a long enough time designing it and you can produce enough of them, you can do extraordinarily complex things very, very cheaply. I mean you see this with all the cellular phones that you buy. I mean cellular phones now are actually give-away devices. You look at them to see what's in them and they're inordinately complicated. You buy a personal computer today and it's 1,000 times more powerful than the biggest computers in the world 30 years ago. You can do everything more cheaply now than you could before.

What's the hitch? Well, you see the hitch when you program a computer, namely, if you look at word processing languages, yes, they become bigger and bigger every year, they become more and more complex, they will do more and more things. As far as their function of word processing, some of us think there have been advances in the last 30 years. Others, like myself, feel that there have not been advances and, in fact, we're going backwards. The problem is we have so much complexity we don't know how to deal with it anymore. Most of us have become blinking 12's. You know what a blinking 12 is. It's all of the electronic equipment you have in your home, whenever you don't program it right you see a 12 blinking on and off, which is where the clock is and the clock hasn't been set, and therefore, it's blinking at you. Most people who have complicated devices, whether they're audio devices or what have you, are using less than one percent of the fancy features in them. You spend hours and hours trying to bring that from one percent up to two percent. So that the cost of complexity is not in what you can build, but it's in this conceptual complexity. If you design an algorithm and you understand the algorithm, it hardly makes any difference how complicated it is, unless the complexity is going up exponentially with something. That's the first caveat. It's actually the second caveat, too.

Complex systems are often not well thought through. They often don't work or are not robust. The non-robustness is the worst part of it. The third caveat is when you

have special applications. Since they involve small numbers, you're not going to build a lot of them, it takes a long time to design them. The only way you can build special applications is to make them minor modifications of other things that have been done before. So this question of how complicated the systems we can build are, if you can make enough of them they become cheap. If you have enough lead time they become cheap. Which says that this layering of communication system that we're talking about really ultimately makes sense.

Starting next time, we're going to start talking about this first part of source coding, which is the discrete part of source coding. We will have those notes on the web shortly. You can look ahead at them, if you would like to, before Monday. Please review the probability that you are supposed to know because you will need it very shortly. Thanks.