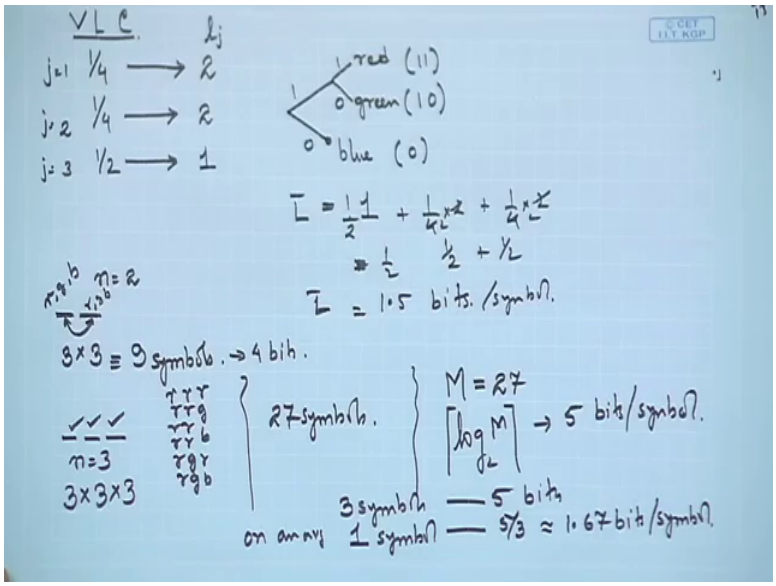
Modern Digital Communication Techniques
Prof. Suvra Sekhar Das
G. S. Sanyal School of Telecommunication
Indian Institute of Technology, Kharagpur

Lecture - 14
Source Coding (Contd.)

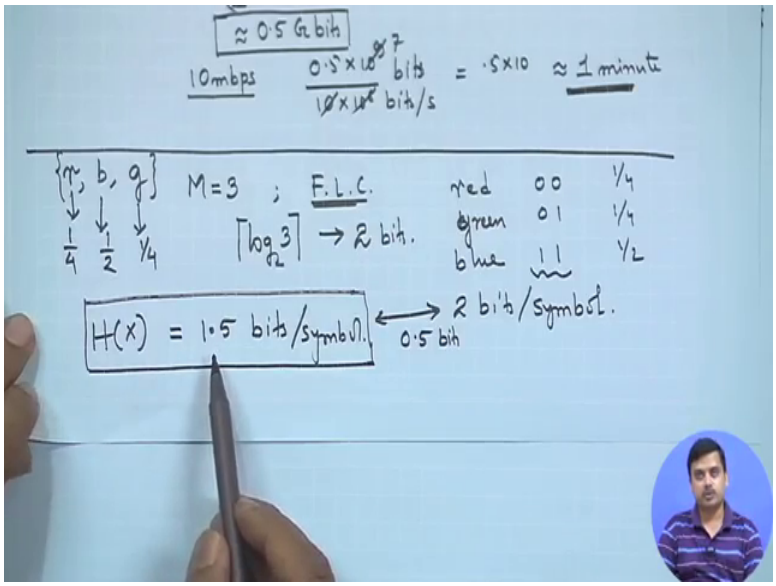
Welcome to the lectures on Modern Digital Communication Techniques. In the previous lecture we have been looking at some examples of Source Coding. And we have taken a particular example, where the three colors in one source represented by red, green, and blue. And we had given their probabilities, so I will just cross check

(Refer Slide Time: 00:45)



So, we had blue with the probability of half, and green and red with probability of one-fourth; so with probability of half, 1 by 4 and 1 by 4. And what we had seen is that if you would do fixed length coding, so here if you do fixed length coding.

(Refer Slide Time: 01:18)



In that case you are going to get 2 bits per symbol that is what we did. And if you would do variable length coding then you could get 1.5 bits per symbol. And the entropy of the source was 1.5 bits per symbol. So, we establish that if you want to do variable length coding you can achieve near entropy rates.

So, what we are going to show you now is that we can also do a source coding with fixed length coding and improve upon the result that you need 2 bits per source symbol. So, let us try and look into that. Now, let us say we take a tuples where n is equal to 2. So, if n is equal to 2 then we have 2 symbols, so each symbol or each place can be filled by 3 different options: that is red, green or blue; again red, green or blue. That means, this position could be filled in 3 different ways, this position could be filled in 3 different ways; resulting in 9 symbols. So, if it results in 9 symbols that means we would require 4 bits. So, we are going to require 4 bits

So, we can move on in this way and what we can summarize over here is that there are 2 symbols and 2 symbols required 4 bits. So, that is 2 symbol tuple n equals to two requires 4 bits. So, on an average you require 2 bits; that means your back to the situation of 2 bits per symbol in this case. That means you are not doing any codes in this case. So, rather if you move ahead and make let us say n equals to 3 as a tuple; that means, 3 symbols.

Again the first place can be filled in 3 different ways, the second place can be filled in 3 different ways, third place can be filled in 3 different ways. That means, I could get a red, red, red, red, red, green, red, red, blue, and then again a red, green, red, red, green, blue and so on and so forth I could keep on doing. That means, I could generate 27 symbols. If I would generate a 27 symbols that means, in our terminology we have the source where M is equal to 27.

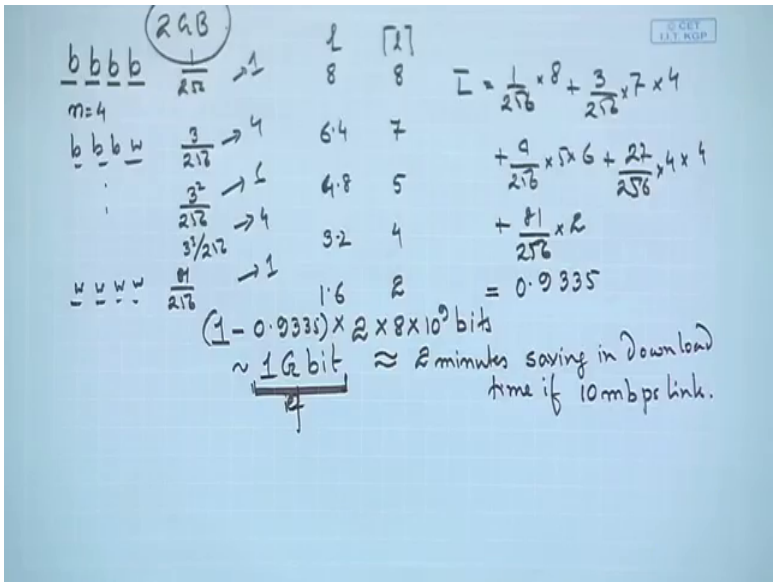
So, if we do this then the number of bits required to encode this would be a ceiling function of this that would result in 5 bits per symbol. Now, if you look at this we have 1 23 symbols requiring 5 bits that means, we could write down 3 symbols require 5 bits. So, on an average 1 symbol will require 5 upon 3 and that is approximately equal to 1.67 bits per symbol.

So, what we have established is that if you can group in tuples even through fixed length coding instead of the previous mechanism by which you are achieving 2 bits per symbol by raw encoding you can improve upon the situation. And you can arrive at a number which is closer to the entropy that is what we had shown earlier and now it has been reduced to 1.67 bits per symbol. Now, these examples would also highlight or bring out to you the meaning of these fractional bits per source symbol. It might be a bit confusing that; what is the meaning of 1.67 bits per source symbol, but this exactly what it means. That on an average 1 symbol requires 1.67 bits.

And when we did variable length coding as is clear here we can clearly see that you can achieve on an average, so this is L bar. So, on an average you can achieve so many bits per source there is 1.5 bits per source symbol. So, this is no confusion about these 0.5 bits which is on average. So, sum of the code word are of length 1 some of the code words are length 2 which are again indicated here. Now along with these probabilities with this code word length it lands up with this value.

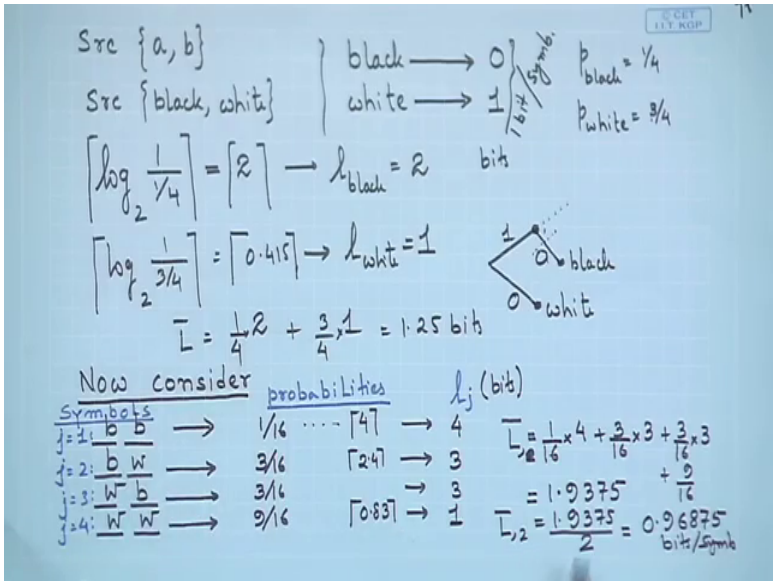
So, I hope this clarifies some of the basic issues that we encounter in source coding.

(Refer Slide Time: 06:30)



Now moving ahead further, we get back to our calculations where we had said that you have a 2 gigabyte file to be transferred. So, we take that particular example and we take another combination instead of taking 2 bits. Now we say that let us take 4 bits now. If you recall that previous example that we had we said that let there will be a source which generates a black and white.

(Refer Slide Time: 07:00)



So, this particular source and then we did some calculations and we found that you can encode the source by taking 2 tuples with 0.96 bits per source symbol. When we did this

we showed that you can actually get a saving of 0.5 gigabits for sending a 2 gb file. And which would also account to almost nearly 1 minute of download time saved.

Now, we get back to that same example and remember in this case we had a black coming with the probability of 1 by 4 and white coming with the probability 3 by 4. So, we get back to this example and these calculations once again. And now we say that what would happen if we take four such source symbols and form a tuple. If you going to do that in that case you are going to have n is equal to 4 and you can get a black or a white, a black or a white, a black or a white, and black or a white; and this black can come with probability of 1 by 4, and white can come with probability of 3 by 4.

See if you are going to do this particular exercise you will find where you want to get from 0 0 0, 0 indicates black and 1 indicates white. So, you can get black, black, black, to let us say a black, black, black, white, and so on and so forth you are going to get all these possibilities. And this is a probability of 1 by 4, this is a probability of 1 by 4 and so on and so forth. So, over all this symbol has a probability of 1 by 256. And this symbol has a probability of 81 upon 256, because this is the probability of three-fourth, three-forth, three-forth, and three-forth.

So, if we form this new symbol that means, there would be 16 possible symbols in these and we would do some fixed length coding. So, what we would find is that l would be 8 for this case and down to 0.6 in this case. And if we have to take ceiling of l right, we are going to get numbers 8 and there would be case when there is only one white this occurs with the probability of 3 by 264, there would be cases of 2 whites that would occur with this probability, 3 whites would occur with probability of 3 cube by 256, and 4 whites would occur with this probability. So, here the value of l is 6.4 of course, this set is 4.8, this is 3.2, and so on and so forth. So, when you do the ceiling of it, it is going to be 7, 5, 4, and 2.

So, now these are the number of bits that we are going to encounter. So, when we say 7 why did we do it? Because they will be 4 such 7 bit sequences; because there 4 such possibilities- white could be in all these 4 different places. And if you have this one that means 2 whites it could occur in 6 possible ways. So, there could be 4 such numbers there could be 6 such numbers again this could be in 4 ways this would be in 1 way this would be in 1 way. So, this adds up to 10, 14, 15, 16, so 16 numbers finally, it adds up to.

Now, if you would calculate L bar in this case you are going to get L bar as 1 upon 256 and you are going to get 8 bits right, plus you are going to take this next number so this will be 3 upon 256 that is the probability. And in this case you are going to get 7 bits and there will be 4 such numbers. So, next we take this one; so this is 9 upon 256 is a probability 5 bits is the length 6 such numbers plus 27 by 256 multiplied by 4 bits multiplied by 4 possibilities plus 81 upon 256. This is the probability of this and you would going to assign 2 bits to this, and there will be only one such possibility.

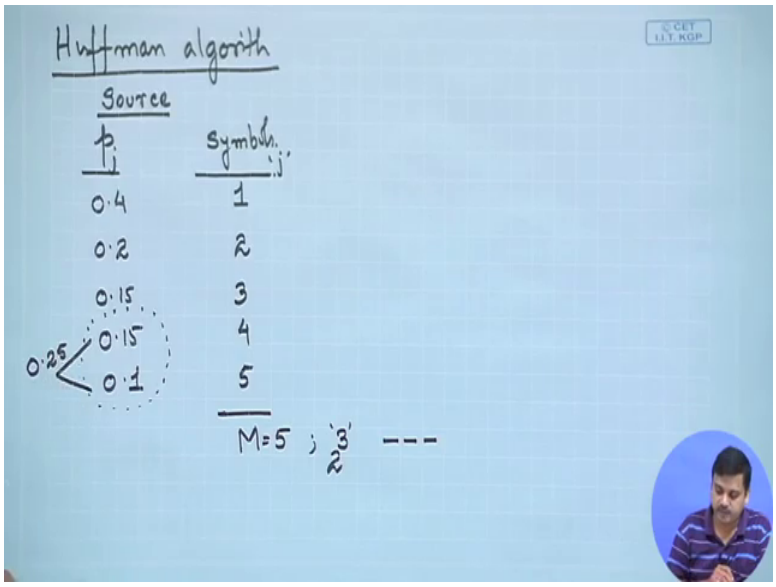
So, when you adds up you end up with the number of 0.9335. If you do this and you try to calculate the amount of savings that you are going to get, what you are going to get is approximately for the same case of 2 gigabyte file you will be approximately getting 1 gigabit of saving, which is roughly indication of 2 minutes of download time if you are 2 minutes saving in download time, if you are using 10 mbps link going by the same method of calculation as we did here; as we did in this particular case. How would you get this answer would be 1 minus 0.9335 multiplied by 2 gigabytes that is converted into bits. And so much of bits so if you are communicating at 10 mbps approximately 2 minutes of time would be saved if you are using 4 bit tuples.

And this is a huge saving if you look at the amount of bits that gets saved. Although this number is not so interesting, but if you would recall you have come one step closer to this number which was the entropy of the source with these probabilities- 0.833. You cannot beat this you cannot at most get down to this entropy. In the previous case when we did 2 tuples you are got down to 0.96 bits, now when you are using 4 tuples y you are getting down to 0.9335. So, slowly as you are increasing the number of tuples you are decreasing it. So, again it establishes the fact that if we make larger and larger tuples you can even with fixed length coding, you can come closer and closer to the entropy. But of course, what we seen if we have use variable length coding it becomes much much easier.

So, with this we come to an end of discussion of source coding using fixed length and variable length coding by the techniques that we have discussed till now. Now, what we see is this particular method requires you to calculate the code word length, it would require you to compute the define the tree, and then you follow up and form these code words map this code words to these particular symbols and then you encode.

Now could there be any other method which is quite efficient as these. So, at that point there were lots of investigations and one of the outcome that is very very useful that happened is the Huffman encoding.

(Refer Slide Time: 14:58)



Or the Huffman algorithm which is used to find code words almost automatically given a particular. So, in this particular part we are going to discuss about how to use this particular algorithm in try to achieve a similar result as we had done before. So, if we consider a source; let us consider a source where the probabilities of the p j's are mentioned and for different symbols.

So, symbol number 1; we are going to index the symbol number 1 and symbol number 2 or basically j you can say. You are not going to write a 1, a 2, a 3 because this is sufficient we simply say that let there be 5 symbol. So, we are taking the case where capital M is equal to 5. And as in the other cases you are given the probabilities, so this let us say 0.4, this is let us say 0.2, this is let us say 0.15, this is 0.15, and this is 0.1. So now, you have given these probabilities and you have to form code as is the typical problem.

So, what we have seen, we have done in the previous cases is if we have to do a fixed length coding the simple step that we did is here we counted M is equal to 5 and this would lead to calculate log base 2 of M and what we would get is number of bits would be 3, that means we require 2 to the power of 3 possible symbols. So, 2 to the power of 3

is greater than 5 so it is not a problem. That means 3 bits, so each symbol would be encoded using 3 bits.

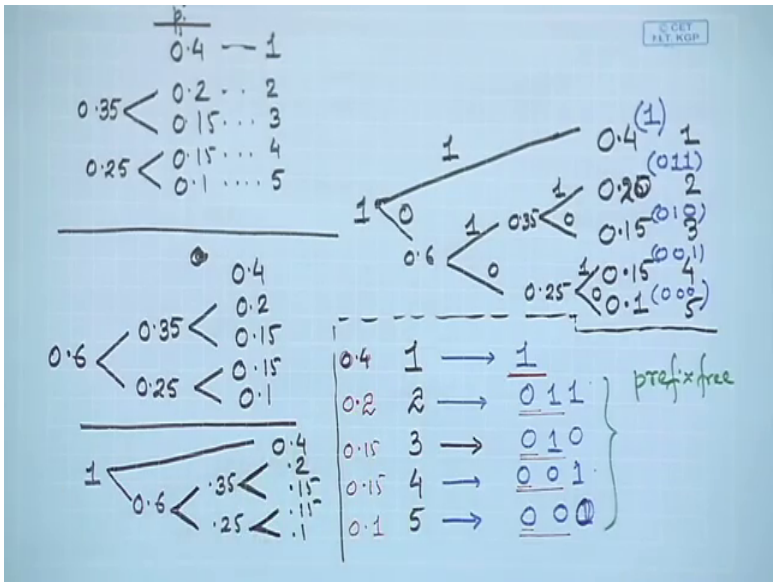
And if you would do a variable length coding you would do log base 2 of 1 upon these numbers, these probabilities you will find the code word lengths, then you would construct the binary tree. And once you constructed the binary tree you would assign the leaf nodes to these particular symbols and then you get your code. So, you are already found the length, and you have match the length with this symbols, and your job is done what those examples we have taken already.

Now instead, in Huffman algorithm things are done in will be different way; so what they do is you would start off by looking at this probabilities and this symbols; it is a algorithm, it is a mechanism which you can write a code and you can easily implement this. So, it starts off by taking the symbols with the lowest probability. In this case we have these symbols. Then it what it does is it forms a reduced set of symbols; that means, instead of taking 5 it forms a reduced set where it combines 4 and 5, and if it combines 4 and 5 then this combination would occur with the probability of 0.25.

So, I will group these two and I would say I have a symbol 1, 2, 3, and a combination of 4, 5. So, 1 occurs with the probability of 0.4, 2 occurs with 0.2, 3 occurs with 0.15 and 4 and 5 together occur with the probability of 0.25.

Now, this process would continue. The next step, in the second let us say or step two what is done is you look at these set of probabilities again.

(Refer Slide Time: 19:08)



So, you have 0.4, 0.2, 0.15, 0.15 and 0.1 and you have already combined these two 0.25. At this point you look at the next two probabilities; next two smallest probability. So, whatever you have done in this particular step you would repeat that, then you would find those 2 symbols with the lowest probability and you would group them. So, here what we find is that 0.25, 0.15, 0.2, and 0.4 these are the new probability sets. So, from this set again if you apply the same methodology we are going to get this as the lowest set of probabilities. So, this was again symbol 1, symbol 2, symbol 3, symbol 4, and symbol 5.

Now we again apply the same protocol or the same method as we applied before. So, what we get? We shifted little bit 0.4, 0.2 and we have combined these two in the earlier steps. Now again we apply the same philosophy what we find is that we have 0.4, 0.35, 0.25, this is the three combined symbols there are left with us. Again we take with the lowest probabilities and we combine them and we form a new symbol, and that occurs with the probability of 0.6.

Finally, in the last step in this particular situation what we get is 0.4, 0.2; so we had combined this two 0.25, we had combined this to 0.35, we had combined this to 0.6, now we are left with 0.4 and 0.6. So, definitely you combine them and you end up with the probability of 1. So, that also means that now all your symbols have been covered and you encapsulate everything into a single symbol.

So far so good, so what do you do with this now you start labeling the branches, you could label the branches in any order in any particular fashion. Since this is a bit cumbersome I will write it a fresh here. This is for symbol 1, for symbol 2, 3, 4, and 5. So, every upward going branch you could label this as 1, every download going branch you could label this as 0, you could do it by vice versa. You could do it this way.

Now if we start reading from this point and go to this probability or to the symbol this string this binary string that we encounter would be the code for this. So, that would mean that we can assign the code 1 for the symbol occurring with probability 0.4. If we travels this branch we are going to get 0 1 1. So, here we can give 0 1 1 to this probability or to this symbol. It follows 0 1 0- 0 1 0 to this probability for this particular symbol. And then we can give 0 0 1 to this or we can give 0 0 0 this particular one.

So, end of the day what we end up with is we have the symbol 1 getting a code of 1, the symbol 2 getting a code of 0 1 1, and symbol 3 getting a code of 0 1 0, symbol 4 getting the code 0 0 1, symbol 5 getting the code 0 0 1. Now if we analyze this code what we find is; so let me draw a margin trying to separate these areas. And once we have this is what we find is that had a 1 which has the probability of 0.4; that means, the highest probability it is getting the lowest code word length. So, the symbol which is having the heights probability getting the lowest code length is as per our earlier description as well.

Two gets has a probability of 0.2, 3 has a probability of 0.15 which is close to that this is the probability 0.15, this is the probability of 1. So, what we can see is that all of these have code word length of 3 and that is quite ok. And this is which is the highest occurring symbol has the lowest occurring symbol code word length.

The second important thing to observe is that 1 is not a prefix of any other code word. We do not find 1, we do not find it here, we do not find it here, we do not find it here, we do not find it here. Now, if we take any other code word let us say 0 1 1. Now first and for most we try to compare you see 1, 2, 3, 4, these are four different code words each having a length of 3. Now since each of length of 3 there could be possibilities of them to be the same, but what we find over here; sorry, I think I have made a mistake at this point it should be 0 0 0 I had make a copy mistake this is the 0 0 0 0 0 0.

So, since all of them are unique and these 3 length that definitely means that none of them are prefix of any other. So, what we end up with this is a prefix free code word.

And what you can verify that whether these particular combinations that we end up with results in the lowest possible code word length if you do by any other mechanism. So, this particular mechanism gives us a direct way of getting into the codes, you are getting into the code word for a particular set of symbols whose probability descriptions are known.

So, at this point I would like to draw your attention to the fact that these techniques that have been explained to you can be used in source coding or source compression. There are many other algorithms, other than Huffman algorithm; for example, the lempel ziv algorithm which you can also find out in many text. Similar techniques are used, I mean not exactly the same there are variations, but running with the same philosophy. If you are compressing a file using let us say a zip or (Refer Time: 28:14) or any other mechanism; similar philosophy would be used.

But try to think over for a minute and see that given any random source; is a pickup a source random, of course the source is going to generate random sequences, but I am picking up a particular file which I do not know a priory. Now if I take two different sources: let us say I pick up one text from a particular subject and another text from novel it is highly likely that the probability distributions of the symbols are going to be different.

So, the question that comes up is what particular probably distributions to be assume a priory. So, this is an open question for you and this particular thing is used in preparing in coming up with better and better algorithms, extending this particular idea. If you look at images there is this jpeg compression and others, they also use a similar philosophy, but the method and things are in details are quite different. The overall objective has been brought out to the particular example where we try to transfer 2 gigabyte file through which we should to you the amount of savings that you can obtain not only in the number of bits that you sent, but also in the download time that you save and which can also be extended to the amount of energy that can be saved by these philosophies.

So, we have almost covered our discussion on encoding of discreet sources. And we will not take up discreet sources any further from this point. In the next few lectures we will be taking up conversion of analog sources to digital form or discreet form. And this particular section of the course that we have covered would definitely get attached to the

output of the section of the analog to digital converter that we will see in the upcoming lectures.

Thank you.