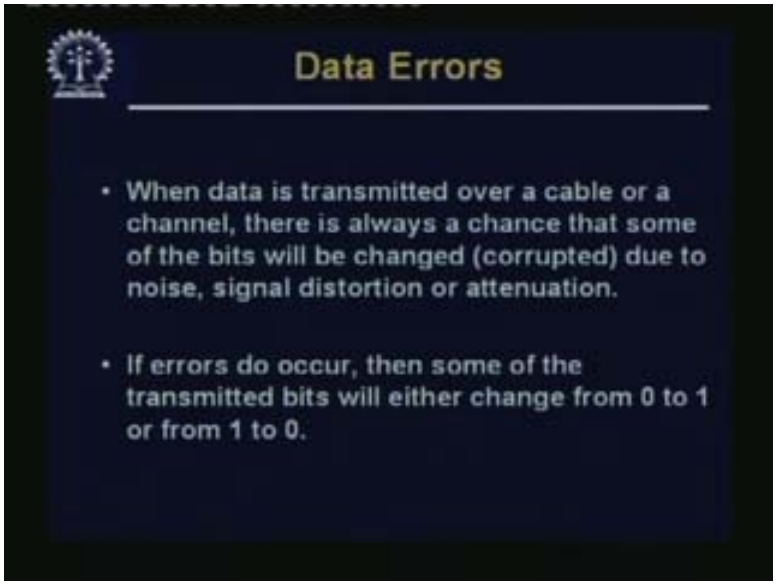
Computer Networks
Prof. Sujoy Ghosh
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
Lecture -16
Error Control

Good day. In the last lecture, we have seen some functionalities of the data link layer namely framing etc. And we have been talking off and on about the error control. Error control means error detection correction, etc. We will be talking about error detection correction in this lecture. We will be talking about error control now.

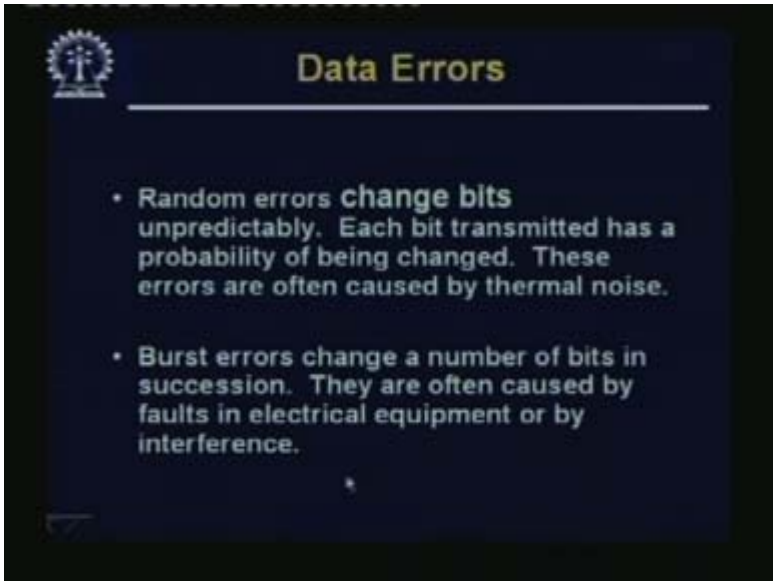
(Refer Slide Time: 01:14)



Now data errors, when data is transmitted over a cable or a channel there is always a chance that some of the bits will be changed or corrupted due to noise, signal distortion or attenuation etc. So many things may happen; for example, suppose you have a wireless channel and suddenly there is a burst of noise, so some of the data will get garbled. Similarly the data may have become much attenuated due to some loose contact somewhere or something and the one that was sent was not received or maybe it was received as a 0 or something.

So, whenever you are sending some data or something, there is some communication going on over some transmission line, you always have to assume that data may not reach the other side in a perfect condition and error control has to deal with how to handle that. So if errors do occur then some of the transmitted bits will either change from 0 to 1 or from 1 to 0. Now what do you mean by error? Some bits were transmitted from 1 and were changed to 0 or from 0 they was changed to 1. So this is the only kind of error that may occur and if you knew where the position of that error is, we can correct it. Because the correct one will be the reverse of the faulty one. So if it is 0, then the correct one is 1 and if it is 1 then the correct one is 0. The whole trick is to find out whether any errors or any change has occurred and if such a change has occurred, where it has occurred and how to handle it. There may be a lot of noise and a lot of data may get corrupted in this fashion. So we will see how or what we can do about this. This data error will also depend on the kind of transmission medium we are using. So that is always there.

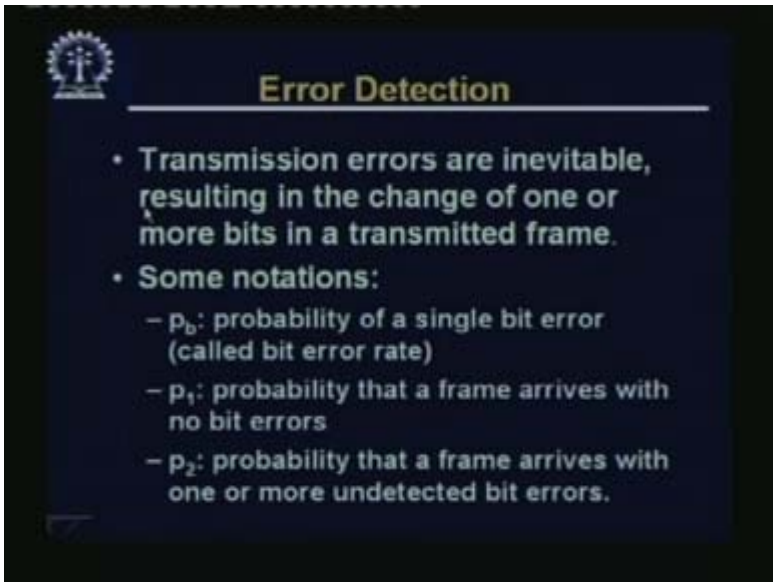
(Refer Slide Time: 03:37)



So, random errors change bits unpredictably. Each bit transmitted has a probability of being changed. These errors are often caused by thermal noise; because this thermal noise is something very general and is always present in any kind of these things. There is higher thermal noise or there may be low level of thermal noise but this thermal noise is always present. So these cause some random errors.

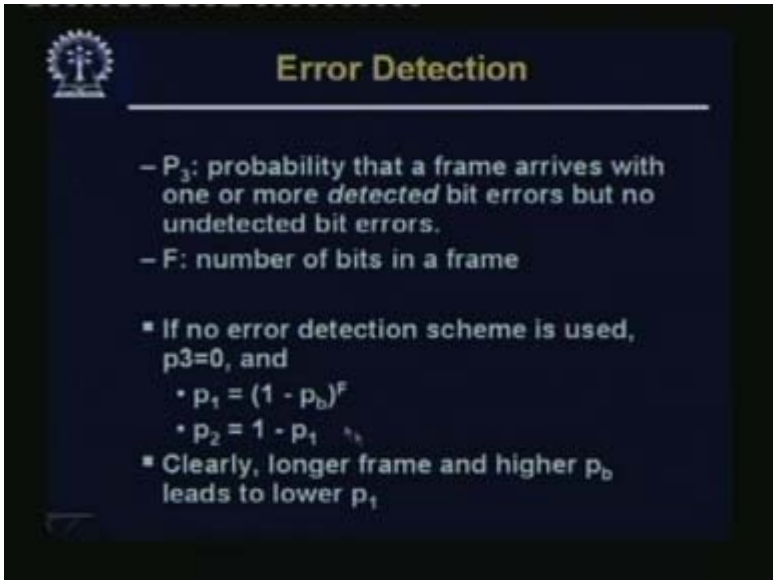
There are some burst errors or a change in a number of bits in succession. They are often caused by faults in electrical equipment or by interference on some neighboring things. So, electrical interference is a very major cause of such burst errors.

(Refer Slide Time: 04:21)



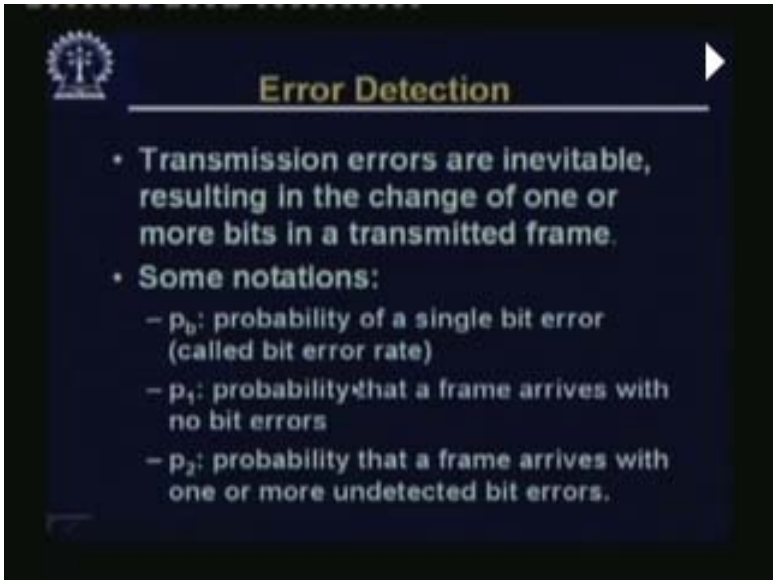
So, we take transmission errors as inevitable, resulting in the change of one or more bits in a transmitted frame. The rate of error may be high or low, but there will always be some error; that is what we assume. Let us just see what it means, if there is a finite probability of error for a particular bit. Let us just have a look at some notations. Let P_b be the probability of a single bit error called bit error rate. So the probability of a single bit error is the bit error rate or b e r as they are sometimes called. P_1 is probability that a frame arrives with no bit errors, that means there is no error. And P_2 is the probability that a frame arrives with one or more undetected bit errors. So, P_1 is no error and P_2 is one or more errors.

(Refer Slide Time: 05:17)

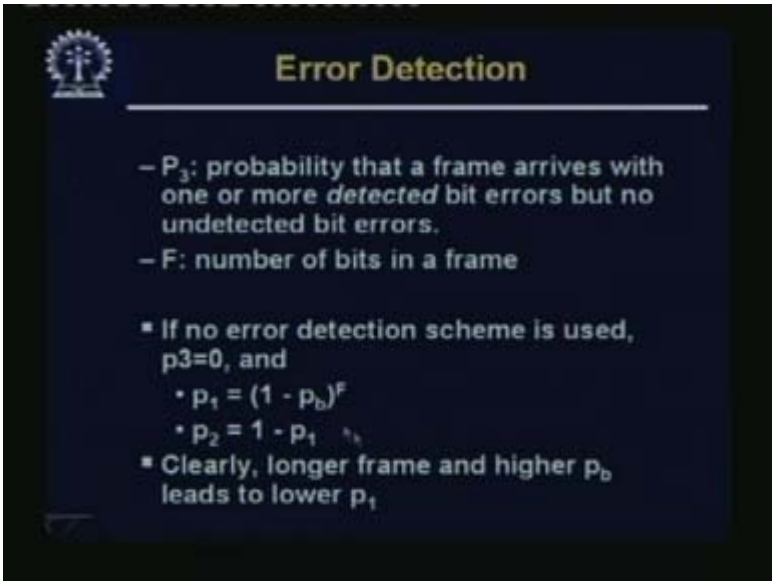


P_3 is a probability that a frame arrives with one or more detected bit errors, but no undetected bit errors. Sometimes we can detect which bits are in error and sometime we cannot. So, we will come to this later. Let F be the number of bits in a frame, if no error detection scheme is used. Suppose we are not detecting it specifically. So P_3 is equal to 0; we cannot detect which bits are in error. So P_1 is equal to $(1 - P_b)^f$. If you remember, P_b is a bit error rate. Since P_b is a bit error rate probability of a single bit error, P_1 would be $1 - P_b$ for one particular bit.

(Refer Slide Time: 05:55)

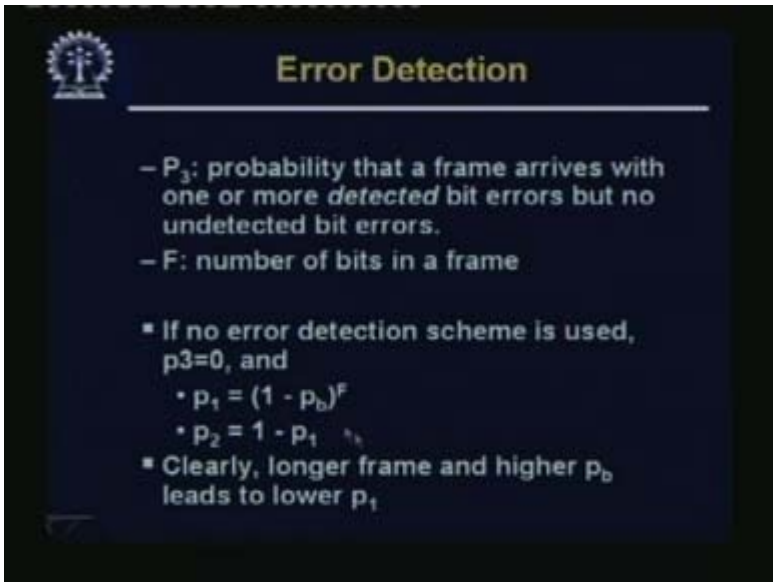


(Refer Slide Time: 06:02)



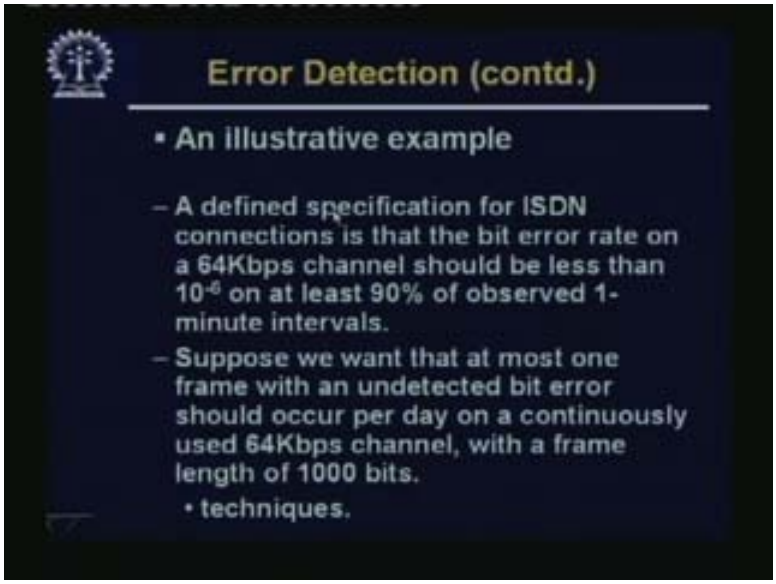
In one particular bit, the probability of error is P_b. So, the probability for that particular bit has no error, that is, $1- P_b$. And we are assuming that these errors are sort of independent, which may not always be the case with burst errors. But for thermal noise, we assume that there are bit errors. We will see that the noise will introduce an error or some other error will creep in. Let us say they are independent. So if there are F bits in a frame, the probability that a frame arrives with no error is $(1 - P_b)^f$.

(Refer Slide Time: 06:47)



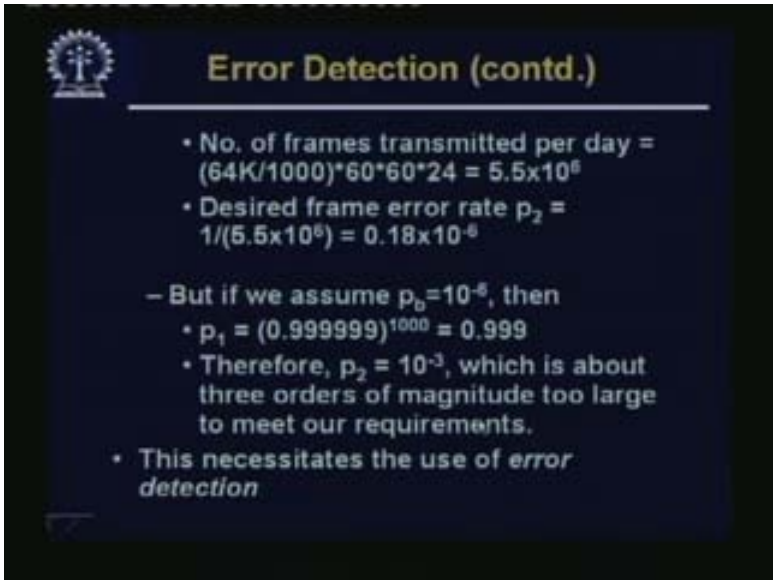
And P_2 arrives with one or more errors, $(1- P_1)$, P2 is equal to $1 - P_1$So, clearly longer frame and higher P_b leads to lower P_1.Let us try to look at some numbers.

(Refer Slide Time: 06:59)



So, this is an illustrative example. A defined specification for ISDN connections – you remember ISDN means integrated service data network which the digital service is given through the telephone lines. So that is the ISDN. So in ISDN connections, bit error rate on a 64-kbps channel should be less than 10^{-6} on at least 90% of the observed 1-minute intervals. Suppose that looks quite low, rate of error is 1 in 10^{16}. That means only in a million bits 1 one bit may come in error. Suppose we want that at most one frame with an undetected bit error should occur per day on a continuously used 64-kbps channel, with a frame length of 1000 bits. So we want not more than one frame should come with undetected bit error. So we are using this for a whole day on the 64-kbps channel and the length of the frame is 1000 bits.

(Refer Slide Time: 08:59)

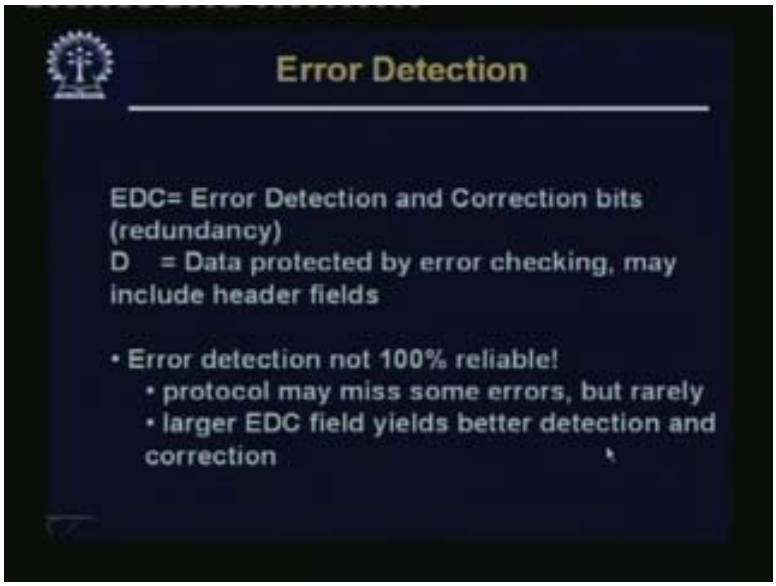


So number of frames transmitted per day is equal to (64K by1000) into 60 into 60 into 24 is equal to 5.5 into 10^6. So if one of them is desired frame error rate, P_2 is equal to1 by (5.5 into $10^{6)}$ ^{is} equal to 0.18 into $10^{-6.}$ But, if we assume $P_{b \text{ is equal to}} 10^{-6}$, then$P_{1 \text{ is equal to}}$ $(0.999999)^{1000 \text{ is equal to}}$ 0.999. So if $P_{1\text{is equal to}} 0.999$, $P_{2 \text{ is equal to}}$ 10^{-3}, which are about three orders of magnitude too large to meet our requirements. That means it is thousand times more, although one bit in a million looks a fairly good error rate. But we find that at the end of the day, we may have a thousand frames which are in error. Well, it does not work out that way that by multiplying with 1000, we will get so many frames in error.

But the point is that whatever probability we require for a faulty frame we have got a much higher probability than we were prepared. We were prepared for say one faulty data frame per day but that is not the case at all. So this necessitates the use of error detection; that means, we have to do something to bring down this error rate. The other point is that sometimes one will not have control over the environment or the transmission line directly. It is quite difficult, because if you are having a wireless transmission. Now, some noise will certainly come from some other source. May be somebody starting a car will also give out some kind of electrical noise and somebody starting a machine somewhere will give some kind of electrical noise, some mixer or grinder running in the house may give some kind of error. All kinds of sources are there, and you do not have any control over it.

So the point is that, whatever the error that is coming, you have to do something at your end to get better error rates. So, we want to detect such errors and if possible we want to correct them. So this is what we are going to discuss now.

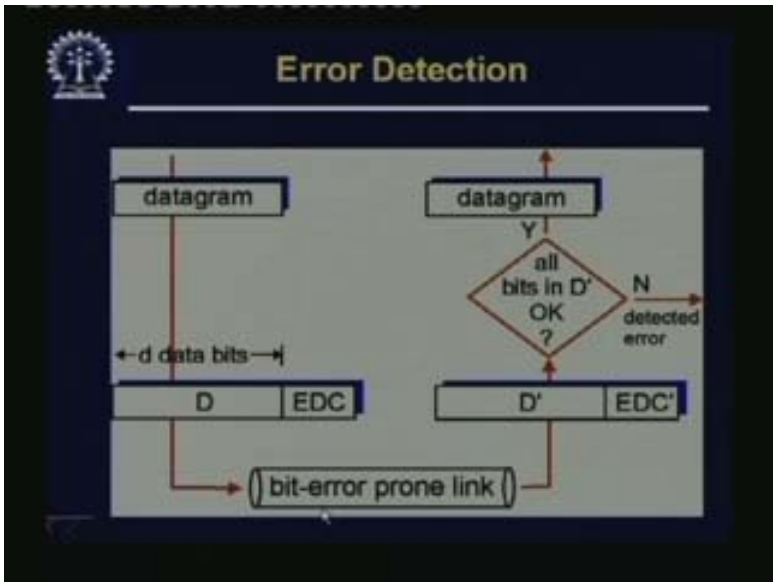
(Refer Slide Time: 11:11)



Error detection: One general principle of getting greater reliability or less error, as I said is always redundancy. That means you put in some kind of redundancy in the system in order to get better reliability. We have seen this when we were talking about the reliability of optical networks. But in this case, the redundancy comes in the form of extra bits that we introduce into the data, which means that whatever data is being sent, we also send in some extra bits specifically for the purpose of error detection and correction. So EDC are the error detection and correction bits which are actually redundant. That means they are not original part of the data, and they are extra bits, which have been introduced.

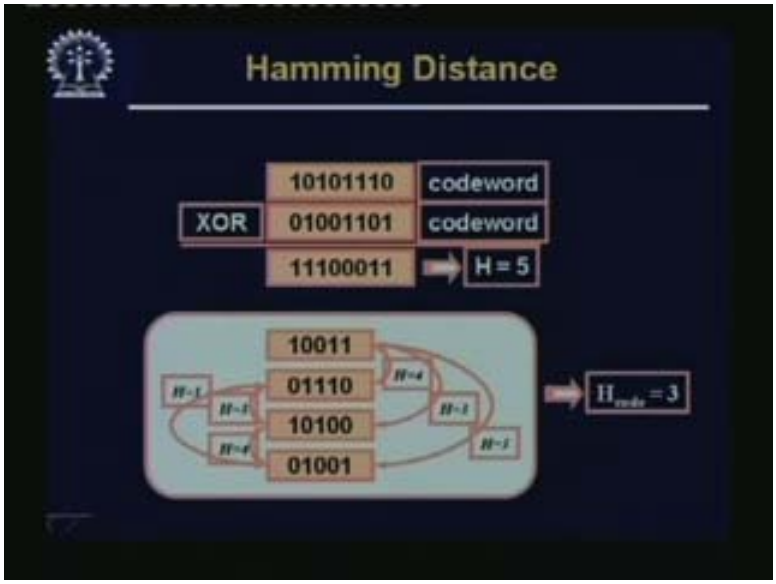
Suppose D is equal to Data protected by error checking; it may include header fields. So the data which we are protecting may or may not include it; it depends on the protocol. It may just be on the payload and the header, all together, we may like to detect the error. Error detection is never 100% reliable. But the protocol may miss some errors but usually we can bring down the level of errors to a very low figure. The larger EDC fields yield better detection. If you put larger redundancy you have better error detection and correction.

(Refer Slide Time: 12:50)



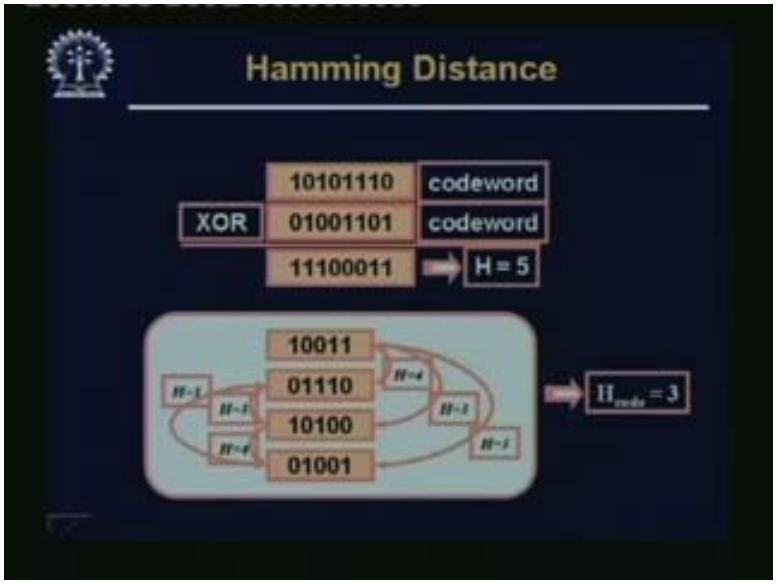
So you have the datagram coming. We add some extra bits EDC from there. We send it to the bit error prone link, which arrives at D' and EDC', now D may be the same as in D' and has no error and it itself may have corrupted or both may have been corrupted. But any way we run some algorithm or some method for finding out whether these are alright and if we think they are alright we say yes. Otherwise, we say that an error has been detected. Sometimes, the errors may be such that although we say everything is ok, it may not be so; in that case, we say that the error has gone undetected.

(Refer Slide Time: 13:42)



Now for this error detection and correction, one concept, which is used, is the so-called hamming distance between the two bit patterns. And then there are hamming codes also. We will not go in to the details of hamming codes here, but let me just tell you what is hamming distance. Suppose we have two code words, like it has been shown here. This one code word is 10101110 and the other one is 0100110, these are the code words. By hamming distance, what we mean is that we have to see in how many bit positions the two code words are different.

(Refer Slide Time: 14:35)



So, in order to find that, we simply XOR the two. So, if We XOR these, we get this. If you can go through 0 1 is 1, 1 0 is 1 and so on. So you get 1 1 1 0 0 0 1 1. Now we count the number of 1s in this XOR, which is 5. So we say that their hamming distance is 5.Please note that if the two corresponding bits are different in the two code words 0 and 1, 1 and 0, then we get a 1 in this XOR. If they are the same like 1 1 or 0 0, we get 0. This H hamming distance is the number of positions where the two code words differ. If you have a set of code words, say between these two hamming distance is 4, between these two, the hamming distance is 3, between these two also the hamming distance is 3.

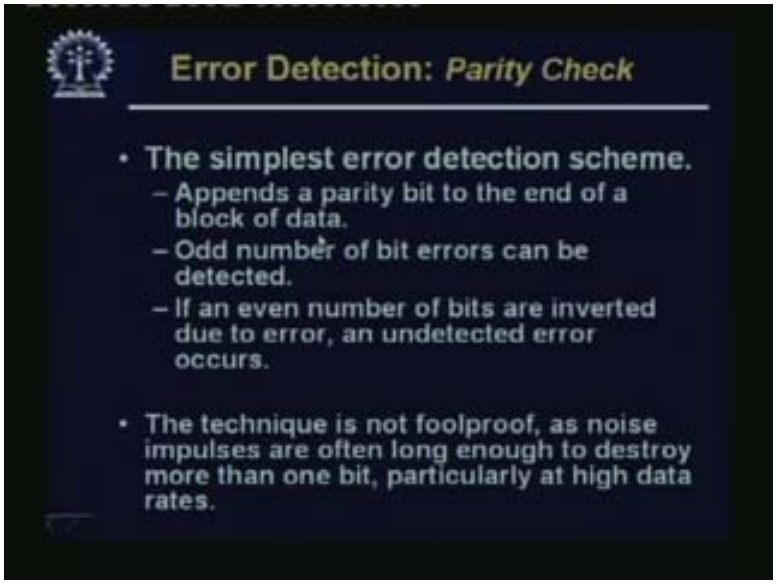
Let us see this differs in one position, two positions, these are the same in the third position. This is again the same. This differs in three positions. Similarly, this other H is equal to 3 for other peers. In this set as a whole, the minimum hamming distance between two code words is actually 3. H of the code word is actually 3. We take the minimum of all these H values and this comes out to be 3. So we say that for this set of code words, the hamming distance is 3. Now, if the hamming distance is 3, we have an interesting situation. Let us say these code words are what we are sending to the other side; now if there is a single error; that means if a single bit has flipped from 1, it has become 0. So what has happened is that this code word will move to another code word.

So this will appear as another code word in the receiving end; and the one which is received and the one which was sent – the hamming distance between the two will be only 1, because only one bit has flipped. So, only one position, what was 0 has become a 1 there, or may be what was 1 here has become 0 there. So it has changed 1 in one position. Assuming that there is a single bit error, the hamming distance between the transmitted and the received code word will be 1. Now if the system is known, then this is the set of code words which is being used on both the sides; and first of all you will immediately know that for this particular set of code words H is equal to 3 was there. That means, say, this is the set of keywords that I showed you; those are the only code words, which had been sent. So the hamming distance was 3.

So for whichever value, this new transmuted or new changed code that was received is not going to be a valid code, because this is only at a hamming distance of 1 from the transmitted code. So this is not going to be a member of the set. So immediately, we will detect that there has been some error. And not only that, if you assume that there is most likelihood that we have got only one error then it is the one which is at a hamming distance of 1 from this transmitted code. It will be at a hamming distance of 1 or 2 from the code, whereas it is at a hamming distance of 1 from the code word which was already sent.

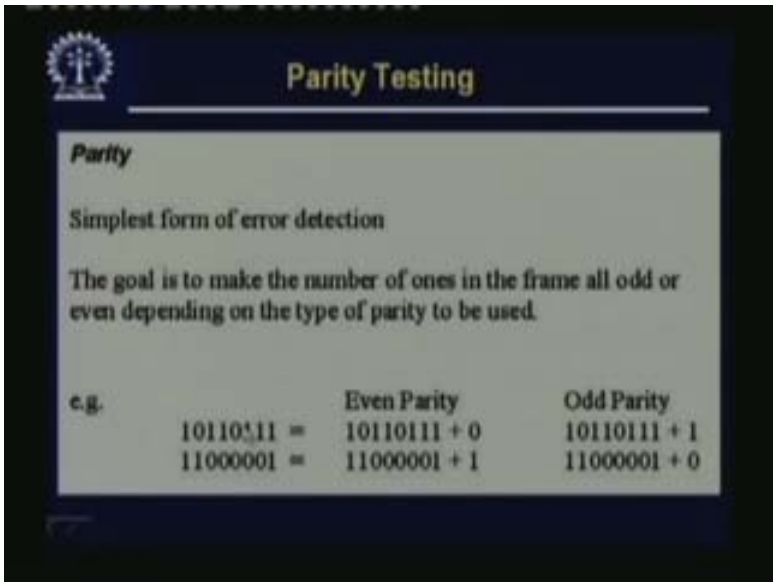
And if you know this and if you assume that there can be only one error then you know that that is the code word. So you can not only detect that there has been an error, you can also correct the error by going to the nearest code word from which the hamming distance is minimum, namely, the one which was transmitted. So this is the general principle. So we will see practically how this is done. There is a coding scheme, for which it goes as hamming code. We will not discuss that, but there are other schemes, but all of them use this idea of a hamming distance.

(Refer Slide Time: 19:08)



The simplest error detection scheme is the parity check. It appends a parity bit to the end of a block of data. What is a parity bit? It is an extra bit, which is sort of added to some data. There are two kinds of parity: odd parity and even parity. Odd number of bit errors can be detected. Why odd number of bit errors can be detected, I will explain that. If an even number of bits is in inverted due to error an undetected error occurs. The technique is not foolproof, as noise impulses are often long enough to destroy more than one bit, particularly at high data rates.

(Refer Slide Time: 19:08)

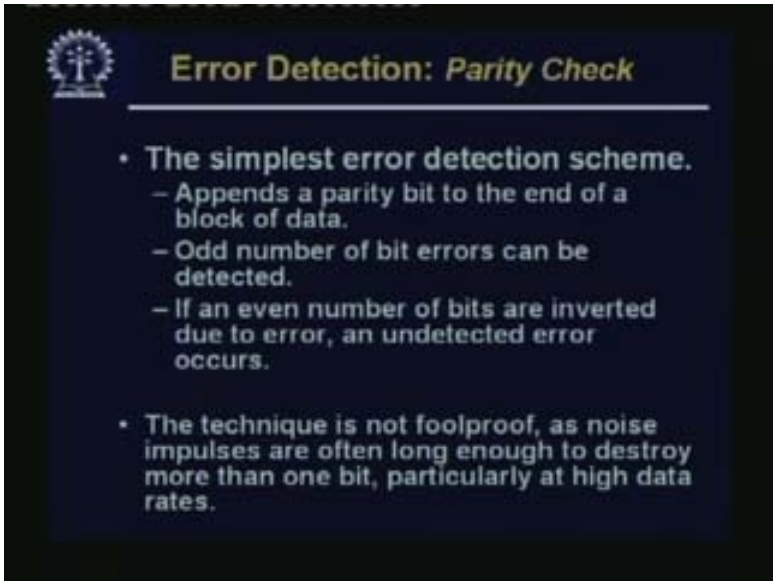


But let us first of all see what a parity bit is Suppose 1 0 1 1 0 1 1 1 was the original bit pattern. In even parity, what we do is that we make the number of 1; we add a bit which is 0 or 1, depending on whether the number of 1s in the original pattern is even or odd. If it is even parity, we want to make the number of 1s even, so the number of 1s were originally even over here. So in 1 0 1 1 0 1 1 1, we have six 1s. So we already have an even number of 1s here, in the added parity bit we just give it a value 0, whereas in the other one, that is, 1 1 0 0 0 0 0 1, we have three 1s. So this was odd, so the added parity bit was also made 1, so that overall the number of 1s in this data plus parity bit taken together is always even.

That is, the number of 1s is always even in even parity if everything is fine; and the number of 1s is always odd. Similarly, the number of 1s is always odd in odd parity. Now if you just think about it a bit, if you are using even parity then all the valid code words have even number of 1s in them. So what is the number of position and what is the minimum number of positions in which they have to differ – these two different code words? Well, the minimum number is 2; that means, the hamming distance of this set of code words will be minimum 2. And if the hamming distance is 2 and if there is a single error, what is going to come is that we are going to get a code word in the receiving end, which has got an odd number of 1s. And, if it has got an odd number of 1s, we immediately know that this could not be a valid code word.

So we know that there has been some error, although we cannot say where that error is, or what the original intended transmitted code is, because if you think of two code words which are at a distance of 2 and the transmitted one – let us say we had even parity, that is, even number of 1s; their hamming distance minimum was 2. Let us take the worst case: the minimum is 2 and then the transmitted one of this code word was this. Where one of the bits flipped, we get something in the middle, which is at the hamming distance of 1 from the two valid code words. So we know that this has been transmitted as invalid. In that sense, this is not a valid code word. So an error has occurred. But I do not know whether something from here came here or the original transmitted code was here and it came here, because we do not know where the error has occurred. So in this particular case, we can nearly detect an error. We cannot correct it at the receiving end. So the case of parity is a very simple kind of scheme.

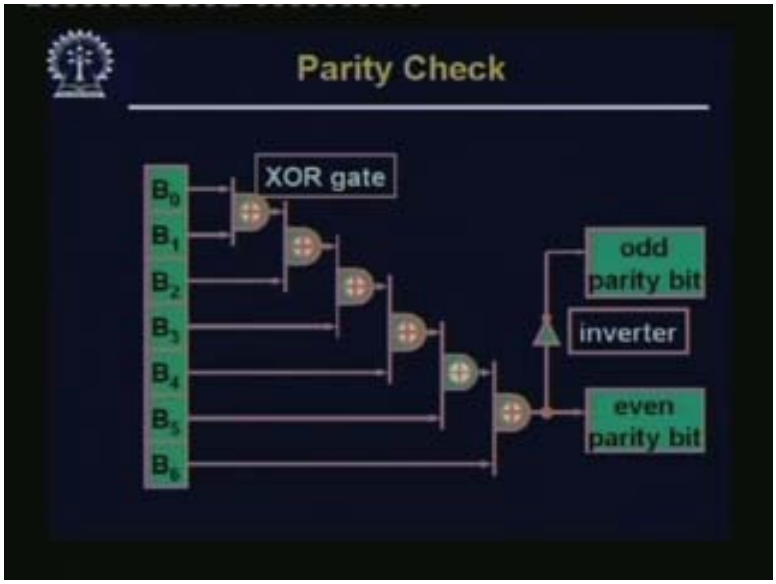
(Refer Slide Time: 23:09)



So once again, if you go back to the earlier slide, I mentioned that odd number of bit errors can be detected. Now as I said if there is a single bit error, it will be detected because now we have got an odd number of points. Well, if there are odd number of errors, that means if there are one error or three errors or five errors in that bit stream, then again, we will get an odd number of 1s on the receiving end. So we can detect that there has been an error. We do not know the number of errors, but in any case, we can detect it. But if there are only two errors, then it will again become an even number of ones on other side which may be a valid code word.

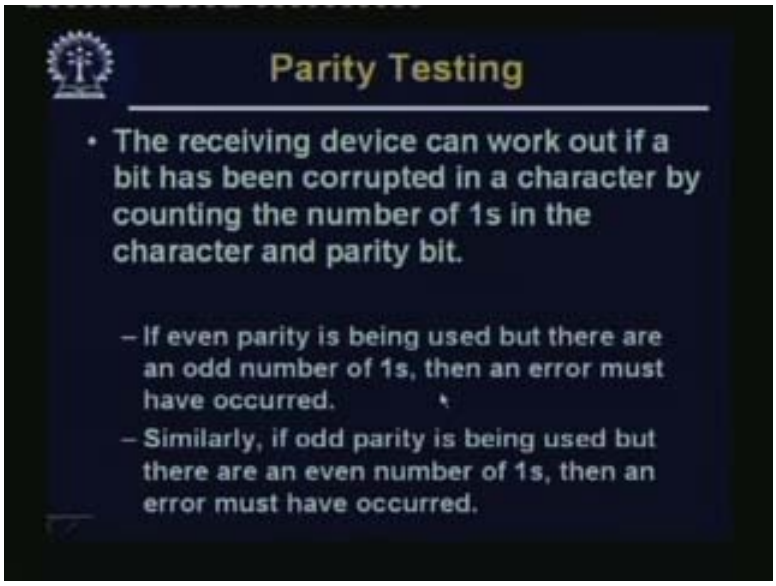
So we will not know whether there has been any error or not. So if there are even numbers of errors, this parity check will go undetected by this single parity bit error checking. Running the parity check is very easy.

(Refer Slide Time: 24:22)

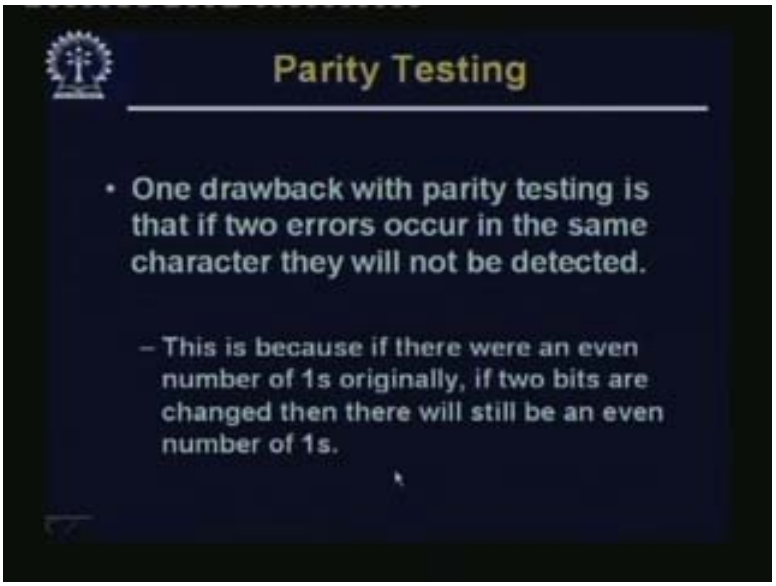


You just run them through some XORs and invert it if you want it in odd parity or if you want an even parity, you can get it and this is very simple. Now parity testing: the receiving device can work out if a bit has been corrupted in a character by counting the number of 1s in the character and parity bit. If even parity bit is being used but there are an odd number of 1s then an error must have occurred. Similarly, if odd parity is being used but there is an even number of 1s then an error must have occurred.

(Refer Slide Time: 24:22)

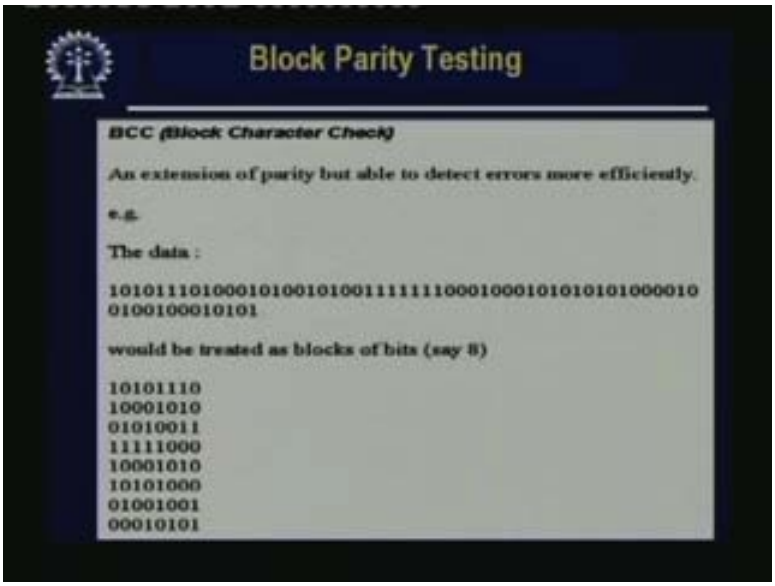


(Refer Slide Time: 24:59)



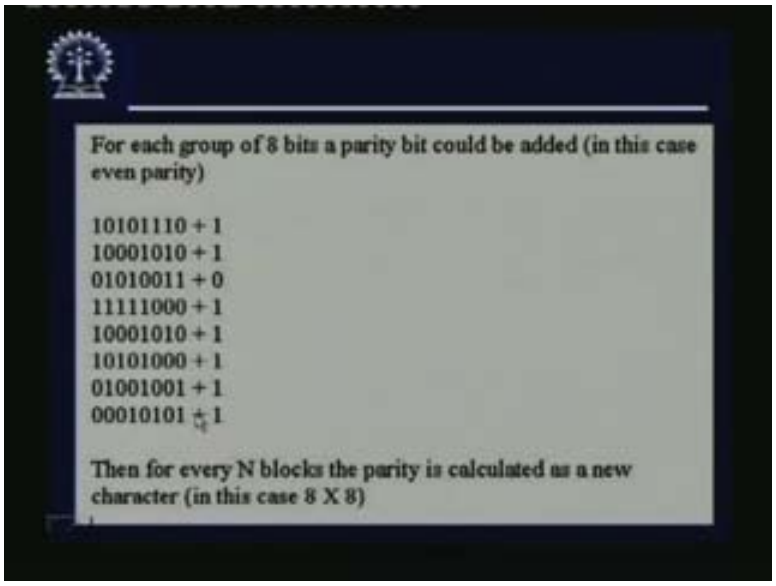
One drawback with parity testing errors is that if two errors occur in the same character, they will not be detected. This is because if there were an even number of 1s originally, if two bits are changed, then there will still be an even number of 1s because either two 1s will become 0. So the even number of 1s remain even or two 0s will become 1, also even number of 0s remain even or 1 1 has become 0 and 1 0 has become 1. So the number of 1s does not change and they still remain even. So it remains undetected.

(Refer Slide Time: 25:37)



There is some improvement over this and we can see how to make it more efficient. This is an extension of parity bit, which is called block parity testing. This is how it works. Suppose we have a long stream of 1s and 0s, 1s and 0s, 1 0 1 1, etc. So what we do is, we break them up into blocks of bits. Let us say 8 bits; we take the first 8 bits as the first pattern. Then the next 8 bits are the next pattern, and so on.

(Refer Slide Time: 26:12)

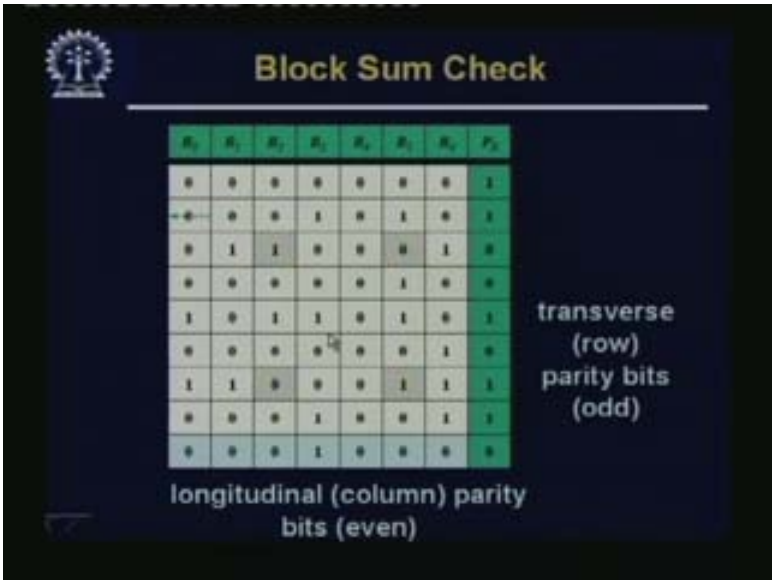


So what we do is, for each group of 8 bits a parity can be added. In this case let us say even parity, so the first 8 bits were 1 0 1 0 1 1 1 0. So for even parity if there are five 1s here so I have to add a 1, there are three 1s here, I have to add 1, there are four 1s here; so I add a 0. In this way for each block of 8 bits, we add a parity bit. Then for every N blocks the parity is calculated as a new character (in this case 8 into 8). So this is how it goes. What we do is, if you look at this as some kind of a matrix. So, we calculate the parity on this side. We calculate the parity along the column also. So five 1s, so we make it a 1 over here for this parity. Then for every N blocks the parity is calculated as a new character. I have mentioned that. I have got parity on this side. So for this block of, say, 64 characters, I have got say 8 plus 8plus1 is equal to 16plus1 is equal to 17 parity bits.

(Refer Slide Time: 27:42)

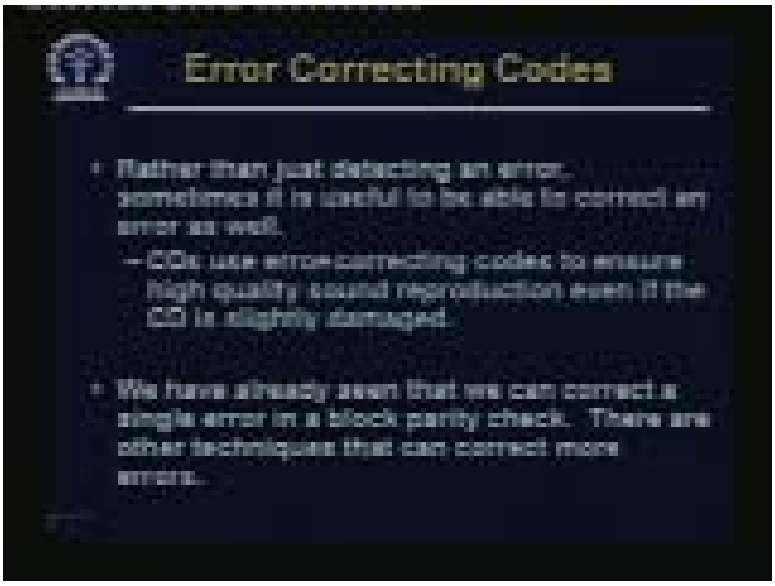
Now if there is a single error, which has occurred, what is going to happen is that where ever it may have occurred in that row the parity test will fail. Similarly in the column also the parity test will fail; and since one column and one row intercept exactly at one cell, we know that that is the bit which has become corrupted. So now if an error occurs, we will know which character and which column it has occurred in. Not only can we detect the error but we can also correct a single bit error by simply changing it. So since we know that this is the bit which is in error, I can simply flip it at the receiving end itself without any referring back to the transmitter. And we know that we have got the correct one. We can not only detect errors but we can also correct single errors in block parity testing.

(Refer Slide Time: 28:43)



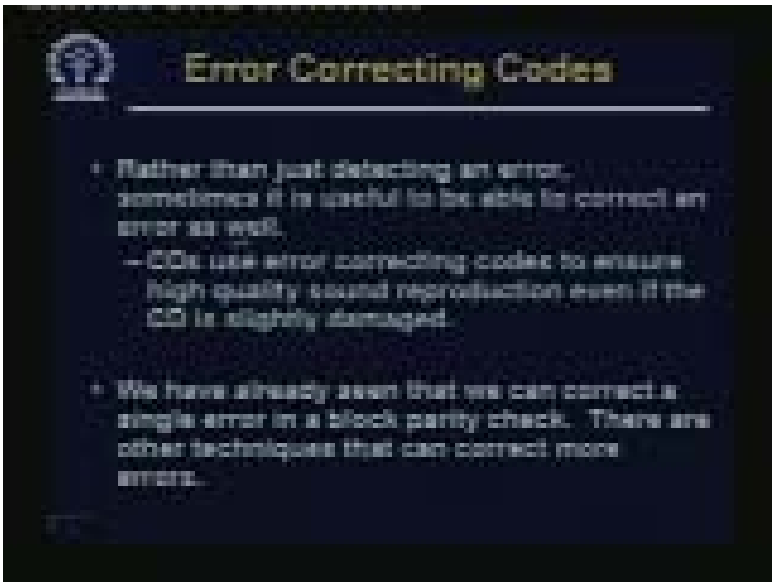
So this is just a better diagram for this; we have got this system. This is using odd parity. I mean there is no reason to use the same kind of parity in both the cases. You can use odd parity for the transverse parity bits and you can use even parity for this and calculate this. So if there is some error we can compute it. If there is more than one error then you may get into some kind of trouble because you will not know which bit is in error. If there are two rows and two columns which show parity error then you will not know which row and which column will match together to get that particular bit, which has been corrupted.

(Refer Slide Time: 29:45)

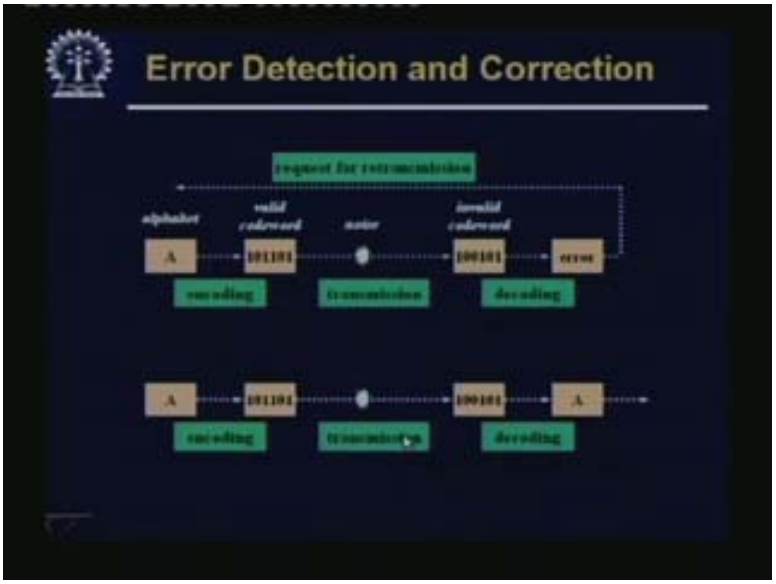


So we not only have error detecting codes but we have some error correcting codes also. By comparing error detection and error correcting codes, error correcting codes require more number of bits because we want to correct it. So we require more number of bits, which is a problem. In error correcting codes, rather than just detecting an error, sometimes it is useful to be able to correct an error as well. For example, CDs use error correcting codes to ensure high-quality sound reproduction even if the CD is slightly damaged. So we are talking about some music CD. So even if a few bits are in error still it will use error correcting code to handle that. We have already seen that we can correct a single error in a block parity check. There are other techniques that can correct more number of errors.

(Refer Slide Time: 30:09)

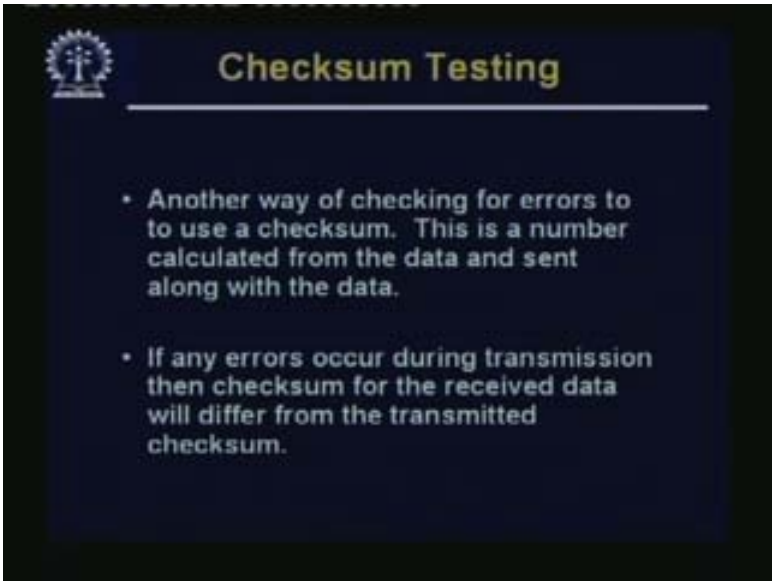


(Refer Slide Time: 30:44)



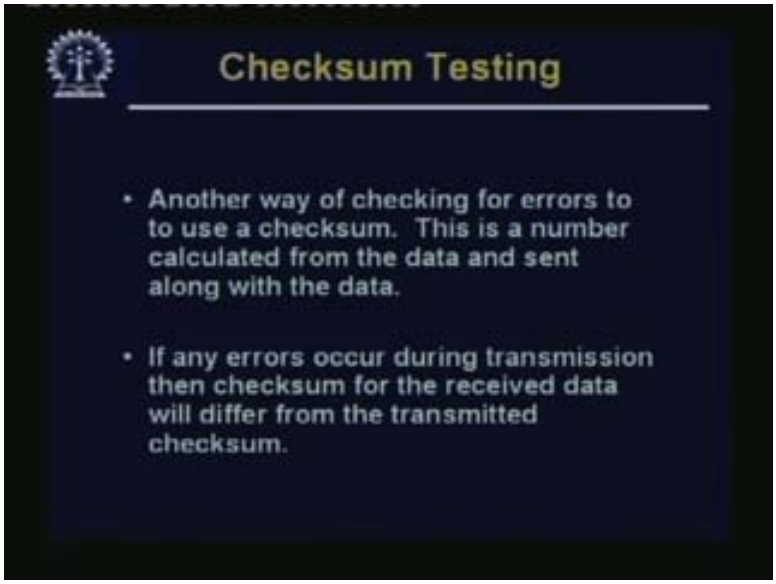
So what happens is that your one scheme could be error detection. So suppose some alphabet A was sent, this was a valid code word, and there was a noisy channel. So some error got introduced. So, 1 0 1 1 0 1 became 1 0 0 1 0 1. That means the third one from the left has become a 0. So this is an invalid code word. We know that the original one was using odd parity kind of thing, so even number and odd number of 1s were there. But in the even number there was even parity and hence even numbers of 1s were there. But I have got an odd number of 1s, I do not know where the error is, so what we do is, we request for retransmission; that could be one approach. But if I can correct a single error on the other side, then I do not need to ask for retransmission. Overall my efficiency may improve but of course for error correction, I have to send more bits. So that is the one area where I am sort of losing something. So it all depends on how you do your engineering and how you come to your particular point of say optimal amount of error correction.

(Refer Slide Time: 32:06)



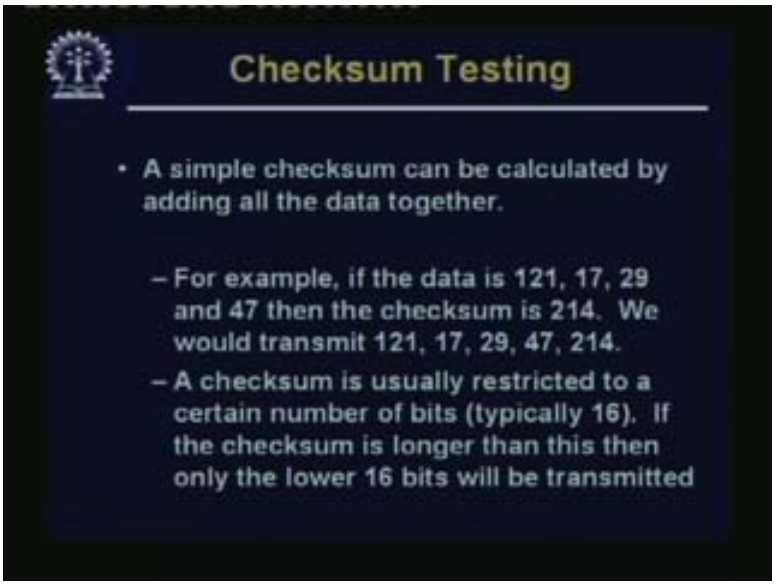
So another way of checking for errors was to use a checksum. This checksum is also used in other fields. So this has been borrowed not just from communication from other fields for accounting also, sometimes checksum is used. This is just used for this sum just for control purpose. So in some arbitrary order you sum it and then with all the data you also check with the sum.

(Refer Slide Time: 32:41)

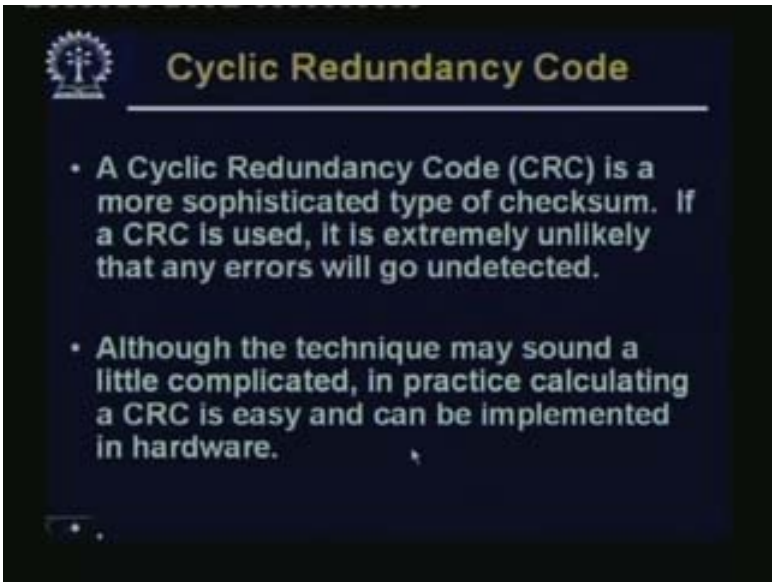


This is a number calculated from the data and sent along with the data. If any errors occur during transmission then checksum for the received data will differ from the transmitted checksum. Of course, the data may arrive alright, but the checksum may become faulty. But even then in that case the checksum and the data will not match. So checksum can only detect errors; it will not be possible to correct it. A simple checksum can be calculated by adding all the data together. For example if the data is 121, 17, 29 and 47, then the checksum (add all the data) will be 214. We could transmit 121, 17, 29, 47 and 214. We know that the last number is the checksum, which should be the sum of all the numbers which have been sent. A checksum is usually restricted to a certain number of bits, typically 16 bits. If the checksum is longer than this, then only the lower 16 bits will be transmitted. But checksum is not very efficient and not widely used.

(Refer Slide Time: 33:06)

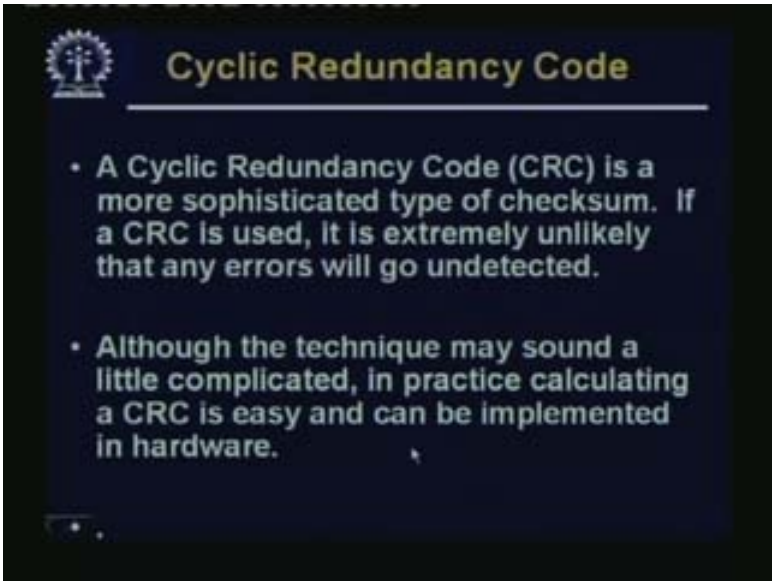


(Refer Slide Tim: 33:46)



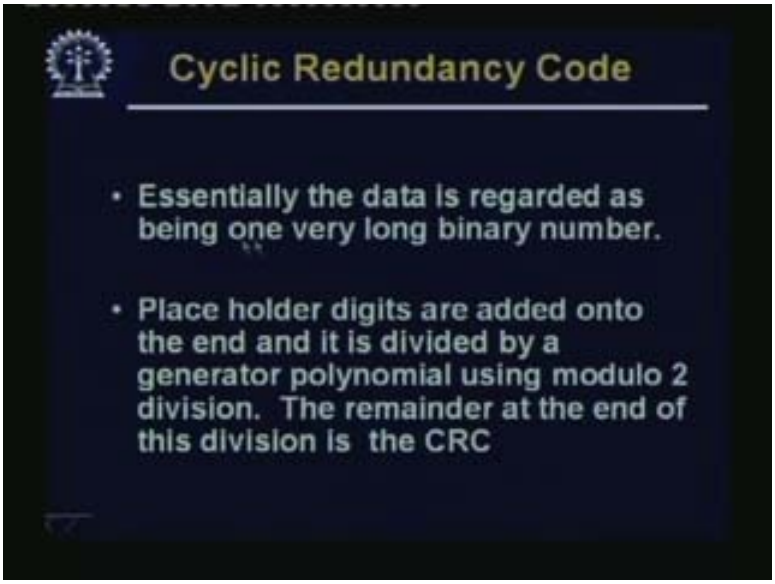
One scheme, which is widely used, is the so-called cyclic redundancy code. What is a cyclic redundancy code? A cyclic redundancy code (CRC) is a more sophisticated type of checksum. If a CRC is used, it is extremely unlikely that any errors will go undetected. A lot of different types of errors are caught by the CRC. Although the technique may sound a little complicated, in practice calculating a CRC is easy and can be implemented in hardware using linear shift registers which is a very simple kind of circuit; through that, we can compute it very easily and quite fast. So, the CRC is preferred in many data link protocols.

(Refer Slide Time: 34:38)

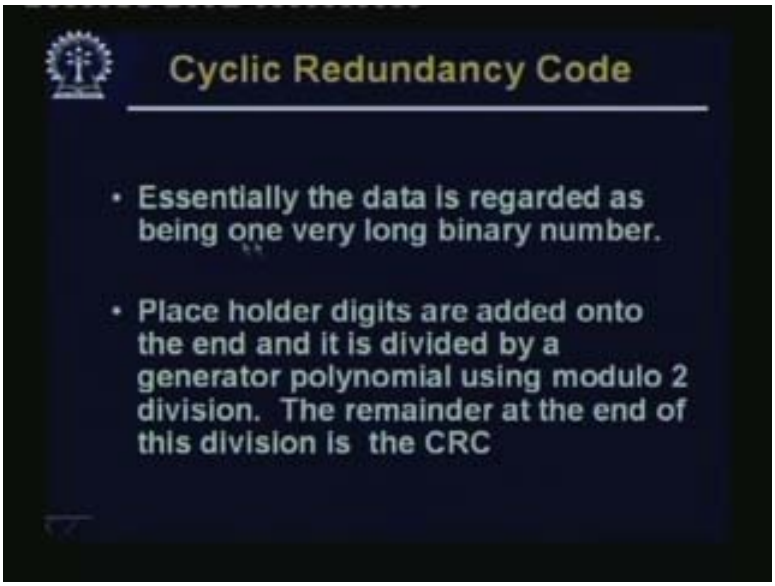


So what is CRC? Let us just look at the details of it. Essentially the data is regarded as being one very long binary number. After all what you are sending is a string of 1s and 0s, and you can take a few of them and just look at it as a binary number. If you are adding the header also, then it got even more mixed up. I mean it was not to be interpreted as a number. But for CRC purpose, we interpret it as a binary number.

(Refer Slide Time: 34:49)

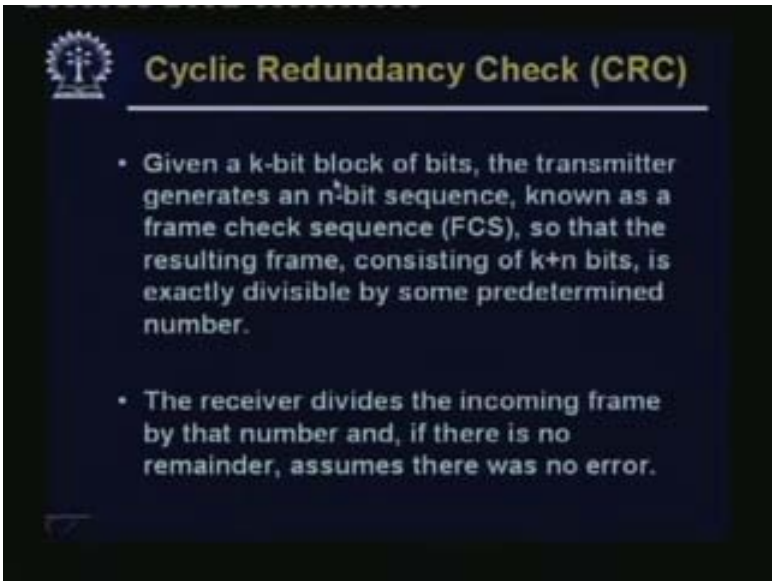


(Refer Slide Time: 35:21)



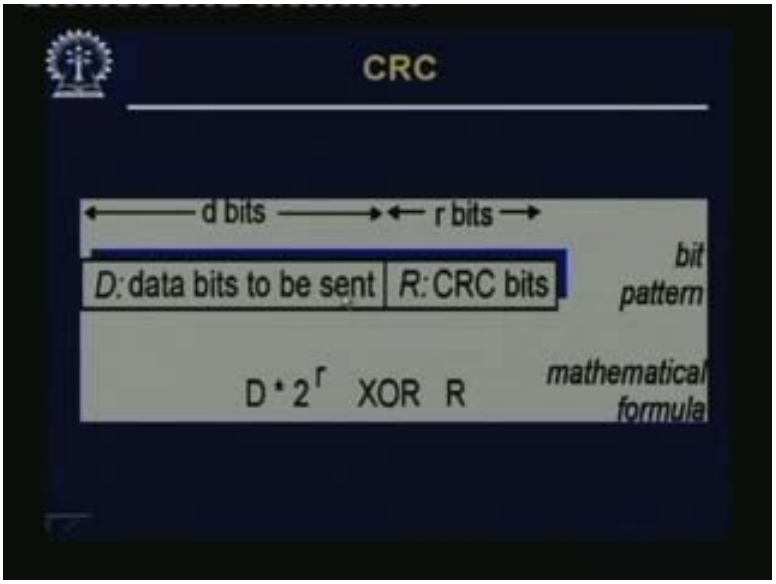
So to this we add some place holder digits at the end and it is divided by a generator polynomial using modulo 2 divisions. So for modulo 2 divisions, you do the same kind of division. Only thing is that if they match then we get a 0, if they do not match we get 1. The remainder at the end of this division is the CRC. So let us just look at this.

(Refer Slide Time: 35:44)

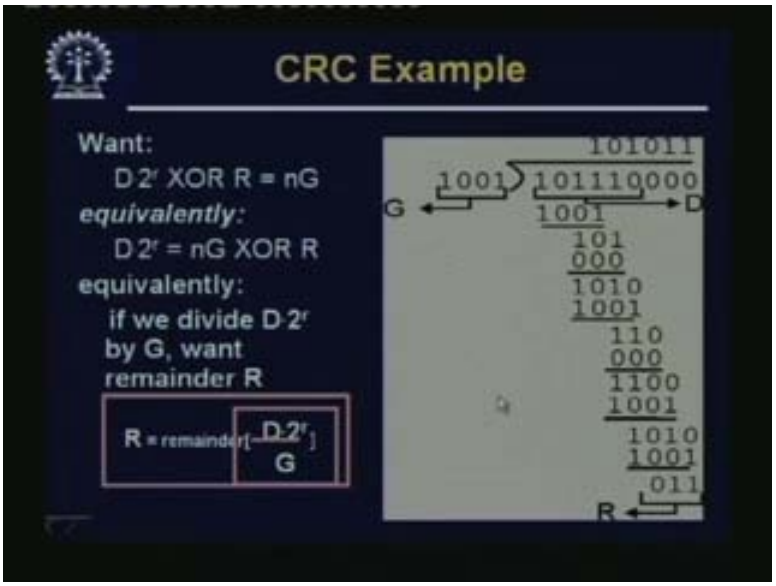


So given a k bit block of bits, the transmitter generates an n bit sequence, known as a frame check sequence (FCS), so that the resulting frame, consisting of k + n bits, is exactly divisible by some predetermined number. The receiver divides the incoming frame by that number and if there is no remainder, assumes there was no error. You look at it as a number; find the CRC; add it to the end; so what will happen is that you will get a longer number. This longer number is supposed to be divisible by the polynomial, which the receiver side knows. So the receiver side will do its own division and if the division is ok, then it will assume that there is no error. If it is not ok, then obviously there is some error. So we have D data bits and R: CRC bits. So how are the CRC bits computed? D * 2^r XOR R. So this is done by some division. So we will see how it is done.

(Refer Slide Time: 36:48)



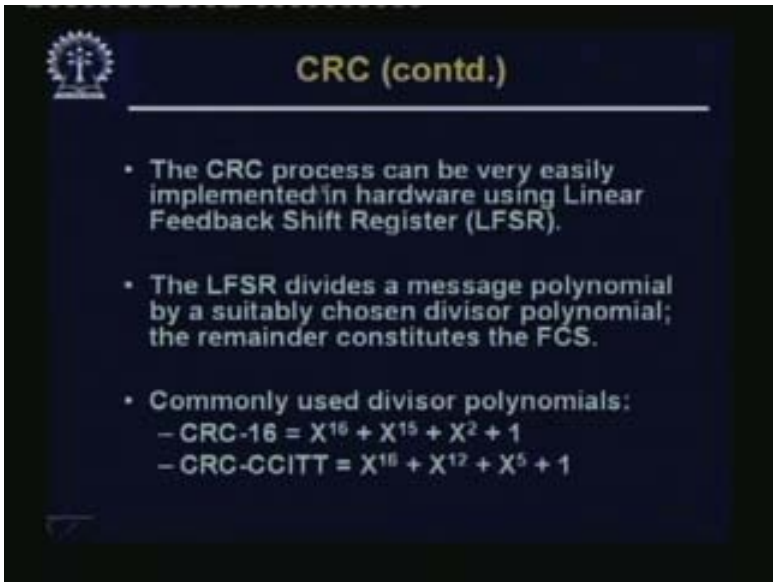
(Refer Slide Time: 37:02)



This is the formula: D into 2^r by G. This is the generated polynomial. Why it is a polynomial – that we will see. Actually the bit pattern is represented as a polynomial; ultimately, this so-called polynomial will be a binary number like this. Suppose the G is 1 0 0 1 and what you are trying to send is 1 0 1 0 1 1. So this is what we are trying to send; and this is the generated polynomial. What we are going to do is that we are going to add some extra bits. So this D into2^r means, adding so many 0s at the end.

So in this case, say, r is equal to 3; so we have added three 0s to the end and I do a division as it is done. So 1 0 0 1 only thing is we do a modulo 2 kind of operation. So 1 1 becomes 0 1 0. As it gives 1, you got a 1 0 1 and so you got a 0 and so on. So in this way the division goes on till we sort of exhaust this and we get 0 1 1 as the remainder. And what will happen is that if you add this 0 1 1 over here, instead of sending 1 0 1 0 1 1, what we do is, we add 1 0 1. Any way, if you add it what will happen is that this will become exactly divisible modulo 2. The modulo 2 divisions will come out exactly.

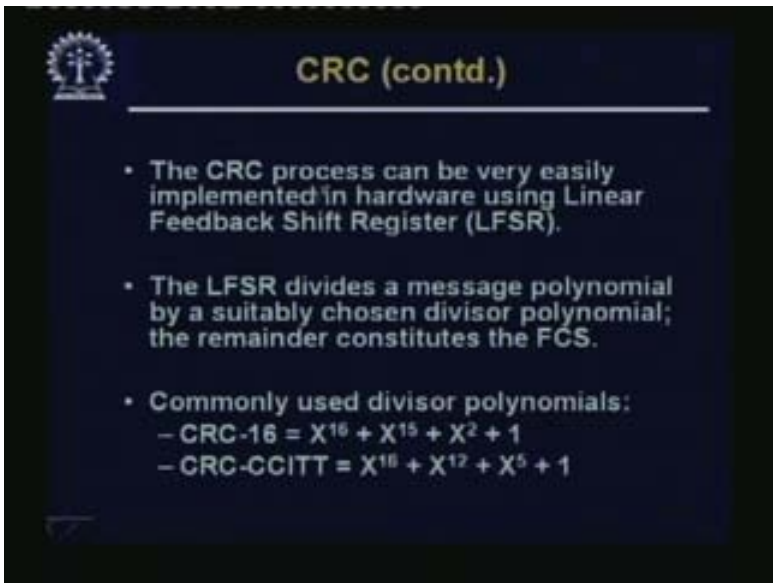
(Refer Slide Time: 38:53)



So the CRC process can be very easily implemented in hardware using (LFSR) Linear Feedback Shift Register. The LFSR divides a message polynomial by a suitably chosen divisor polynomial; the remainder constitutes the FCS, which is added to the data bits.

Commonly used divisor polynomials are: CRC 16 is equal to $X^{16 \text{ plus}}$ X $^{15 \text{ plus}}$ X $^{2 \text{ plus}}$ 1.So actually it is a 16 bit binary number as I said the G is called as generator polynomial. So G will have one in the sixteenth position, one in the fifteen position, one in the second position, and one in the first, that is, 0th position, which means that there are going to be four 1s in this G and all the rest are going to be 0. So how it is generated polynomial? So we are not going into that, how this particular polynomial, why was this particular polynomial chosen rather than some other polynomial. But as we have seen, they cover a lot of different types of errors. Previously if you remember, in our parity bit we were just getting odd numbers of errors. If there are an odd number of errors then we can detect it here and we can detect a lot of different cases of error. So it is extremely unlikely that if the CRC is alright, then there has been some error. So there are different protocols and different systems with different generator polynomial. So CRC CCITT is equal to X^{16} plus X^{12} plus $X^{5 \text{ plus}}$ 1 and so on.

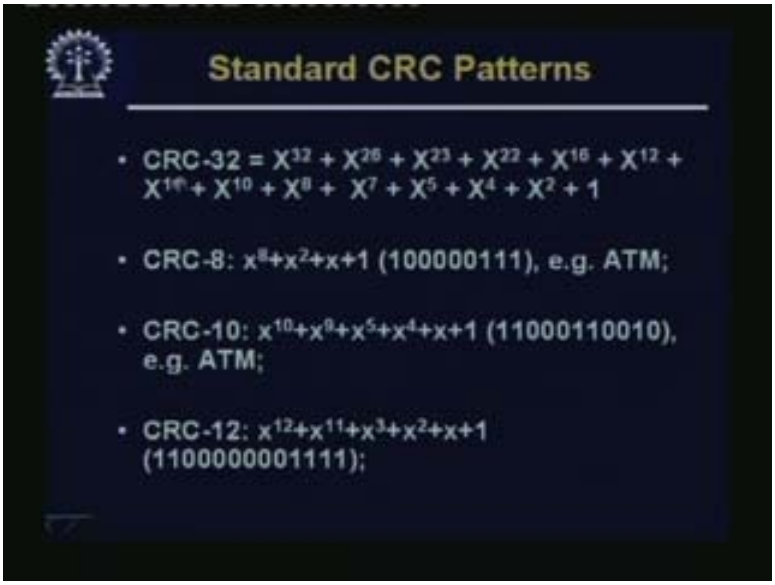
(Refer Slide Time: 40:57)



From the above slide we can see CRC 32 and CRC 8. So CRC 8 means this is an 8-bit thing. This is the eighth position, second position, first position, and zeroth position are all 1s, rest are 0s. So this is used in ATM. CRC 10: this may also be used; and CRC 12. So these are the different CRCs.

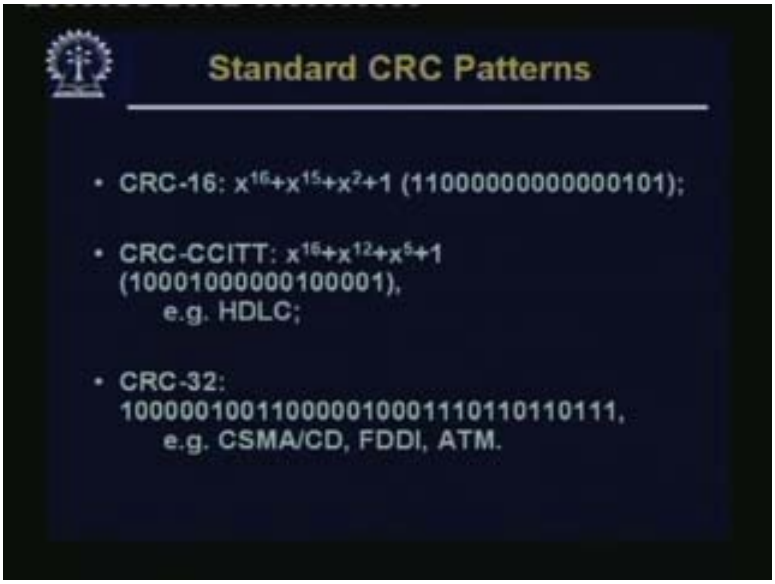
Actually not all polynomials will give you the same kind of error coverage; that is known, and people have found out some properties which are good properties to have in a generator polynomial and then people have found out some good polynomials which will cover a lot of errors. So HDLC uses some kind of errors.

(Refer Slide Time: 41:03)



So CSMA/CD, FDDI, and ATM – they use CRC 32 and so on. I am just showing you, but you need not remember any of these.

(Refer Slide Time: 41:59)



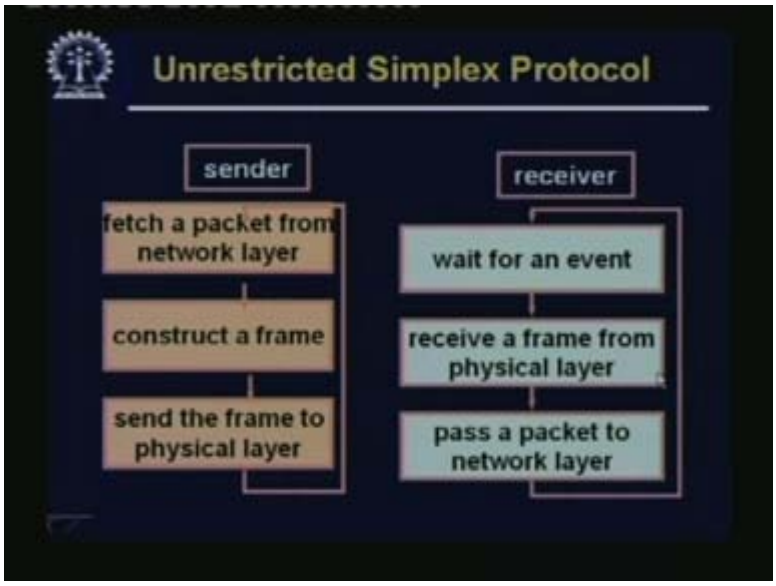
(Refer Slide Time: 41:59)

Now, we come to a slightly higher level, data link protocols. We are still talking about errors and error control, but the point is that, we have been talking about this parity and CRC that is error control at a lower levels. We say a lower level because it is a small amount of error. We can possibly handle it using such error detection or error correction kind of schemes. But what might happen is, because of a very vast noise or something it may garble the data so badly, that it is entirely unrecognizable on the other side. Even if you have an error correction scheme, because none of the error correction and error detection schemes are 100% infallible. So there is still a possibility that there will be some undetected errors and then there will be some data which becomes so bad that there is no question of any error detection or correction. We require some higher-level schemes also in addition to whatever we have talked about this parity CRC etc. So we will now see higher layer data link protocols.

(Refer Slide Time: 43:34)

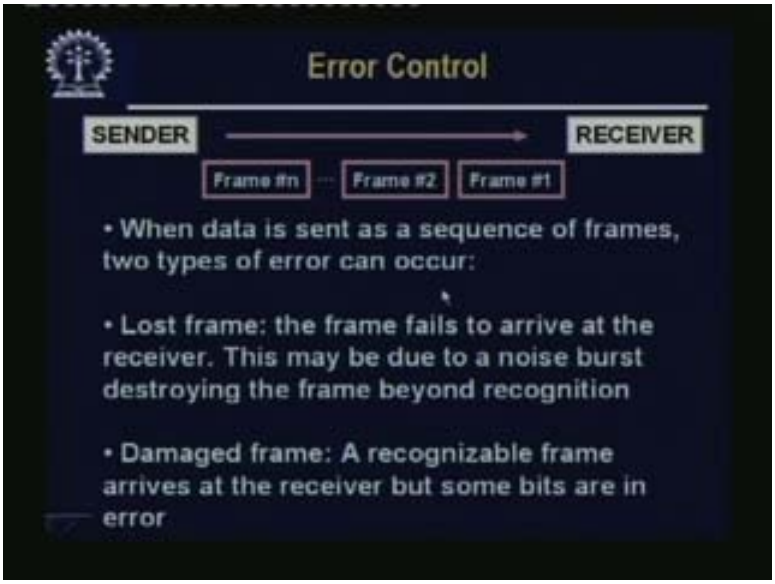
Once again, data link protocols could be of different types; for example, unrestricted simplex protocol: this does not do much as we will see; stop and wait protocol; stop and wait ARQ for a noisy channel means that ARQ stands for acknowledgement request. That means an acknowledgement is expected. One bit sliding window protocol: this is similar to stop and wait, sliding window with go back N ARQ, sliding window with selective reject ARQ, and so on.

(Refer Slide Time: 44:07)

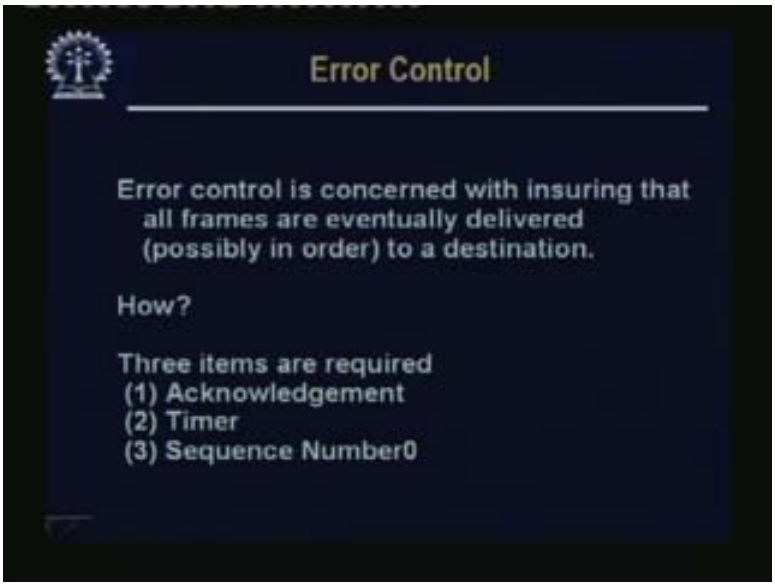


The first is Unrestricted Simplex Protocol. Well, it simply sends, it does not do any kind of error control, and so this is the simplest situation. We have a sender to fetch a packet from a network layer, construct a frame; and send the frame to the physical layer. So the sender is just going on sending without bothering about what is happening on the receiver end. So if things are alright, the receiver waits for an event, receives a frame from physical layer and passes the packet to network layer. The faulty things, which have gone in, maybe it is the responsibility of higher layer systems to take care of that. But the higher layer application does not bother, if a few bits are bad say in the case of a voice channel.

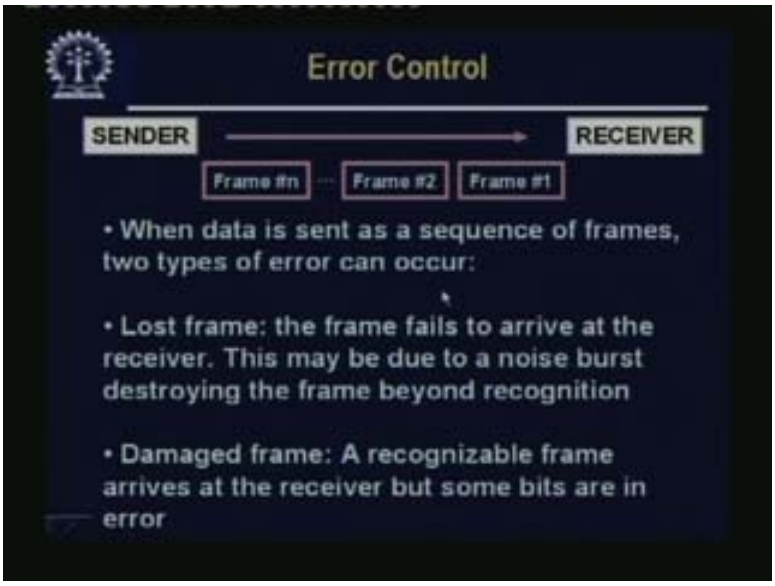
(Refer Slide Time: 44:58)



(Refer Slide Time: 45:03)

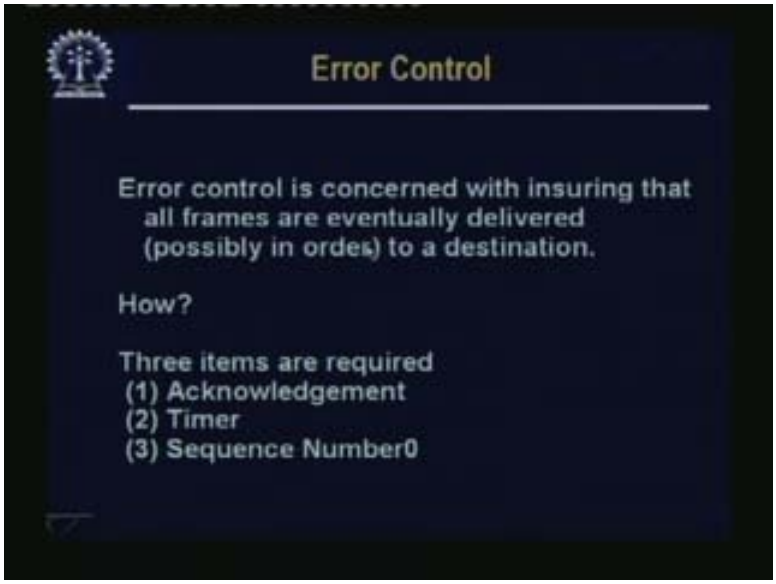


(Refer Slide Time: 45:05)

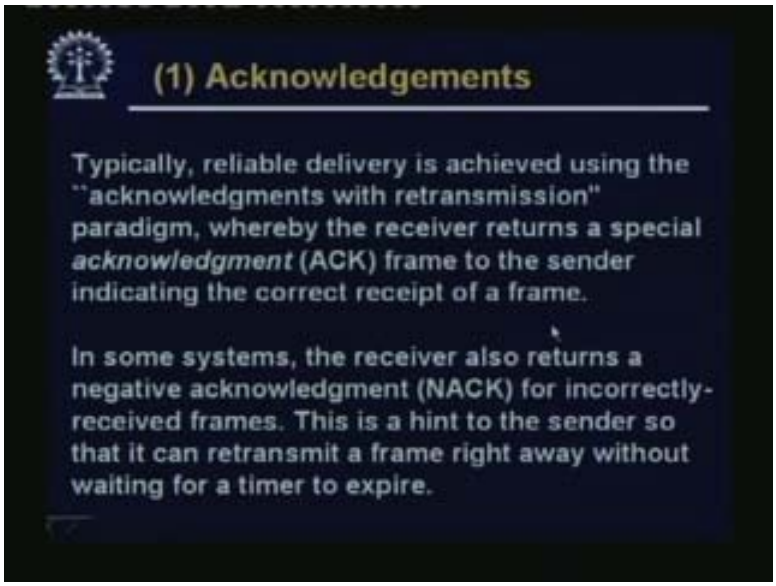


When data is sent as a sequence of frames, two types of errors can occur: One is lost frame: the frame has been lost entirely and the frame fails to arrive at the receiver. This may be due to a noise burst destroying the frame beyond recognition. It may have a recognizable frame arriving at the receiver but some bits are in error. A frame may be damaged or a frame may be entirely lost. See sender and the receiver. The sender is sending a number of frames one, two, to frame n. Frame one is going first. So the point is that some of the frames in between may get lost all together. We have to make the worst case assumptions. Error control is concerned with insuring that all frames are eventually delivered (possibly in order) to a destination. So, how? Three items required are acknowledgment, timer and sequence number.

(Refer Slide Time: 45:58)

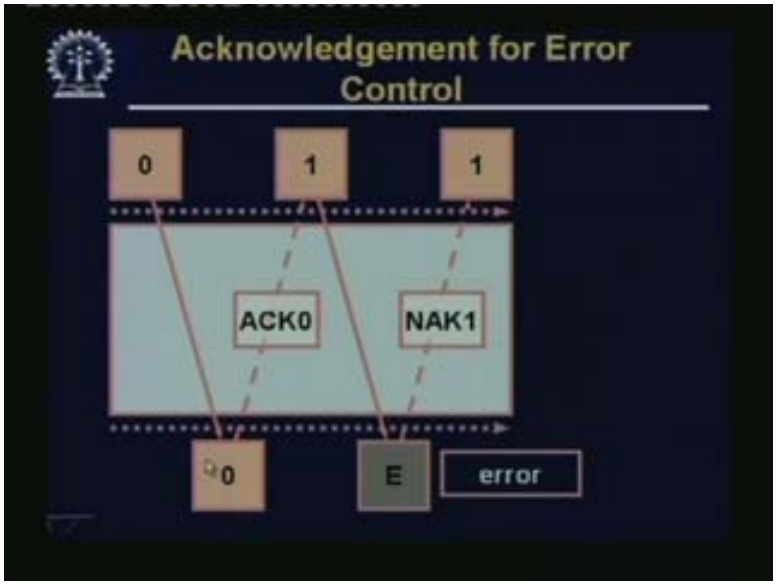


(Refer Slide Time: 46:12)



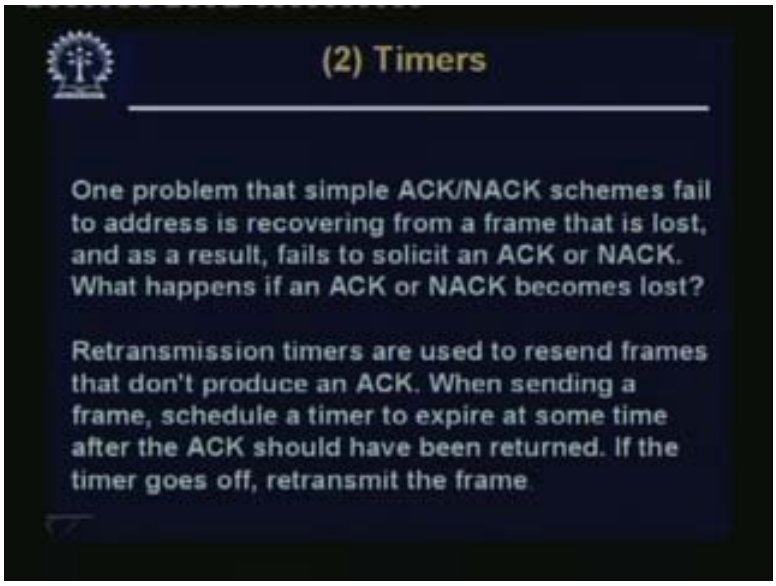
Acknowledgment – what do we do with acknowledgment? Typically reliable delivery is achieved using the "acknowledgments with retransmission" paradigm, whereby the receiver returns a special acknowledgment (ACK) frame to the sender indicating the correct receipt of a frame; that means the receiver, after receiving the same will send an acknowledgment. In some systems, the receiver also returns a negative acknowledgment for incorrectly received frame. So this is a hint to the sender, so that it can retransmit a frame right away without waiting for a timer to expire. We will come to this timer to expire. The point is that there is a two way communication. The sender is sending the frames, and the receiver is sending the acknowledgments. If it gets a damaged frame, then it will send a negative acknowledgment. Of course, if it does not get any frame, then it will not know whether anything was sent at all. In that case, it may not send any acknowledgment or negative acknowledgment anything at all.

(Refer Slide Time: 47:20)



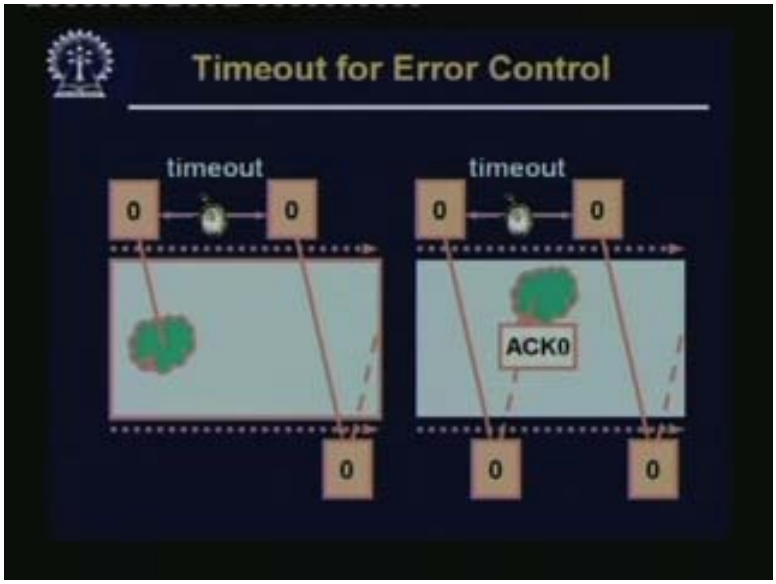
The situation is something like this: suppose 0 was sent by the sender and it arrived. The receiver sent an acknowledgment 0 then a 1 was sent. Suppose I am just taking this bit by bit, just to make it simpler. Suppose 1 was sent it came with some error, we do not know about what kind of error it is. So the frame number 0 was acknowledged, frame number 1 was received, but it was damaged, so a negative acknowledgment was sent. That means, frame number 1 was not rightly received, so frame number 1 is going to be retransmitted by the sender. This is the argument.

(Refer Slide Time: 47:20)



The other thing we require is a timer. One problem simple acknowledgment, negative acknowledgment schemes fail to address is recovering from a frame that is totally lost, and as a result, fails to solicit an acknowledgment or a negative acknowledgment. What happens if an acknowledgment or negative acknowledgment becomes lost? Retransmission timers are used to resend frames that do not produce an acknowledgement. When sending a frame, schedule a timer to expire at some time after the acknowledgment should have been returned. If the timer goes off, retransmit the frame.

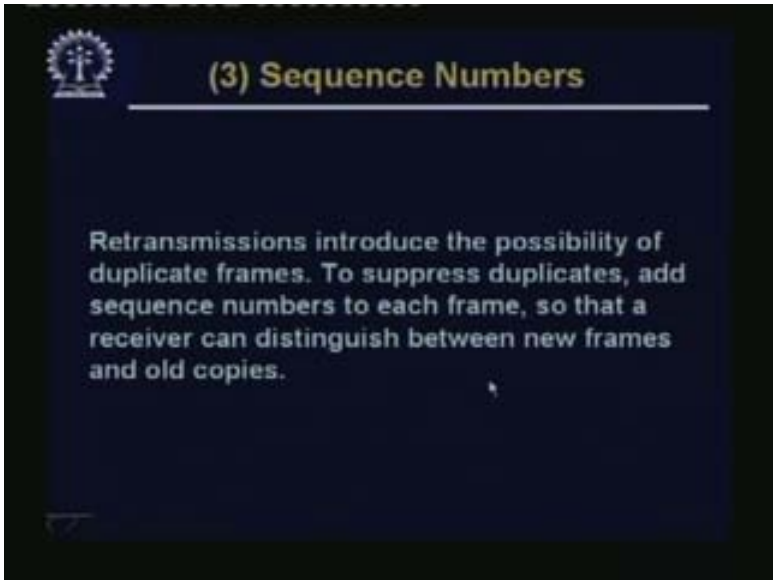
(Refer Slide Time: 48:09)



So this is the scheme, you see a frame which was sent was lost. So the receiver of course did not know anything. What happens is that as soon as the frame was sent, at the same time a clock was started in the sender's end. I have to assume that the system has transmission time, and this is the time it might require for the receiver to process it and send an acknowledgment. Then the acknowledgment will take some time to reach the sender. So, all that is taken into the account with some amount of time thrown in, and that is set in the timer. So if the timer goes out that means, if there is a time out, now the sender knows that the frame must have been lost, otherwise an acknowledgment would have come. So he sends the frame again.

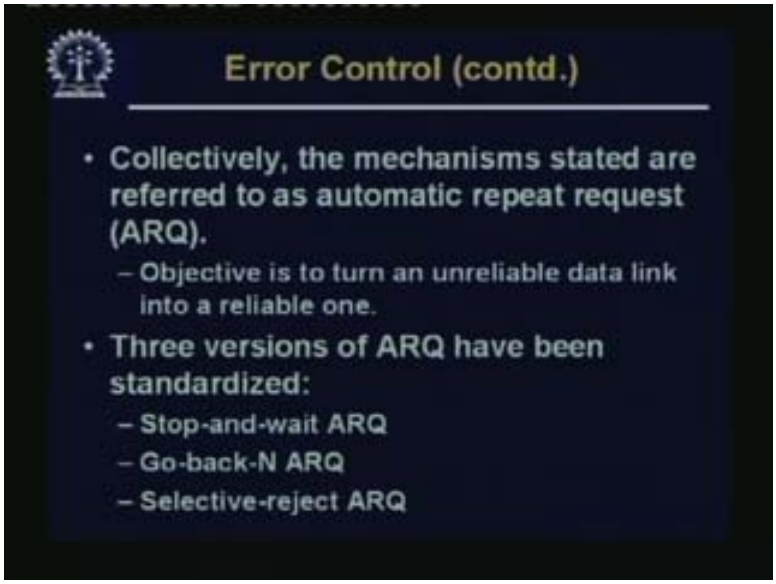
The other thing that might happen is that suppose the frame was sent the 0 th frame was sent it was received and then an acknowledgment was sent, then the acknowledgment was lost. He would not know whether reached at all or the acknowledgement was lost. So after a time out, he will send this frame again. So this is a case where the frame has been duplicated. Depending on what kind of protocol we are using the receiver may or may not understand that this frame is actually a duplicate of this. Maybe two such exactly similar frames were supposed to come one after the other because it is after all coming from some application, which is absolutely unknown to the data link layer. So the frame may get duplicated. But then there are some protocols which take care of that where if the receiving side will know that this is actually a duplicated frame. We will come to that protocol later on.

(Refer Slide Time: 50:51)

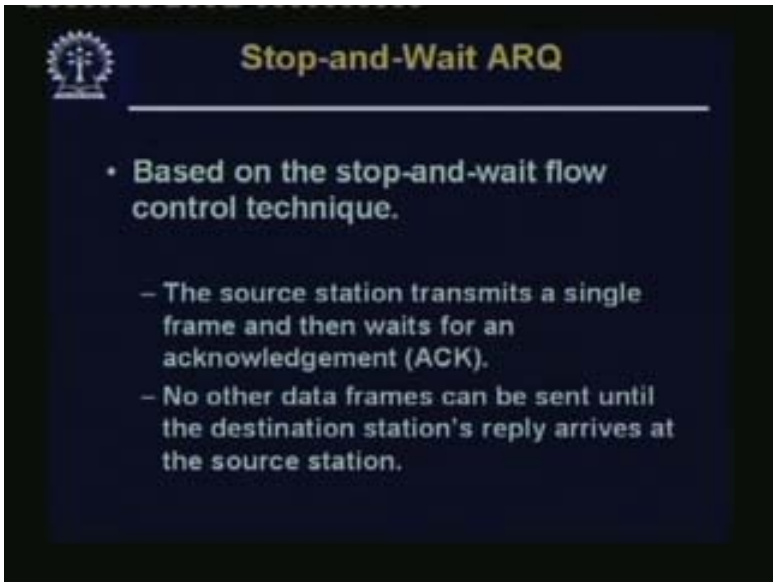


And the third technique that we use is the so-called sequence number. So retransmission introduces the possibility of duplicate frames. To suppress duplicates, add sequence number to each frame, so that a receiver can distinguish between new frames and old copies. So that if the frames are numbered sequentially like say 0, 1, 2 etc. as I was showing you the case where the receiver gets two frames which have the same number 0, it knows that acknowledgment must have got lost or something like that happened. The sender sends the same frame twice, so he will discard just one of them. So collectively, the mechanisms stated are referred to as automatic repeat request (ARQ). The objective is to turn an unreliable data link into a reliable one. Three versions of ARQ have been standardized: stop and wait ARQ, Go back N ARQ, and Selective Reject ARQ.

(Refer Slide Time: 51:40)

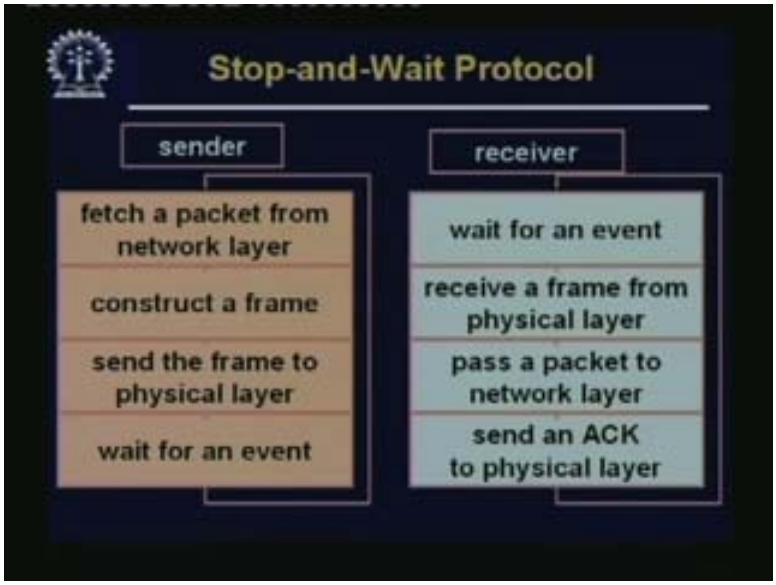


(Refer Slide Time: 52:02)



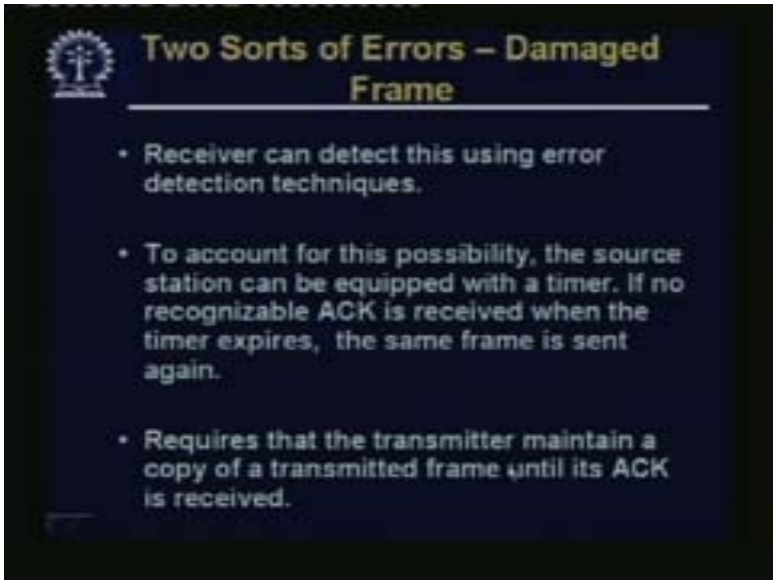
So we have the stop and wait ARQ, which is based on the stop and wait flow control technique. The source station transmits a single frame and then waits for an acknowledgment to arrive. No other data frames can be sent until the destination stations reply arrives at the source station. So it waits for the acknowledgment; if the acknowledgment does not come it will resend after a automatic repeat request. So it will automatically repeat the frame.

(Refer Slide Time: 52:40)



So this is the scheme, we have the sender which fetches a packet from network layer, construct a frame, send the frame to physical layer, wait for an event. So if it gets an acknowledgment then it is fine then it is going to send a second frame. If it does not get an acknowledgment it will send the same frame again. So until it gets an acknowledgment or the timer runs out, the frame which was already sent that has to be stored locally in some buffer in whichever system is handling this data link control. And the receiver side also waits for an event, receive a frame from the physical layer, pass a packet to the network layer, send an acknowledgment to the physical layer. So these is the stop and wait protocol. This is a very simple protocol. Unfortunately, this is not very efficient as we will see that later.

(Refer Slide Time: 53:30)



And so we will continue our discussion on this error control and then we will move on to flow control in the next lecture. So this is a simple kind of protocol but this has got efficiency problem, in the sense that this is not very efficient. So we will continue this discussion in the next lecture. Thank you.

Preview of next Lecture

Computer Networks
Prof. Sujoy Ghosh
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
Lecture - 17
Stop and Wait Protocol

Good day, so in the last lecture we were discussing about stop and wait protocol .we have seen what it is. It is really a simple kind of protocol which is used both for error control as well as for flow control in some cases. But since this is a simple protocol we will look at its performance now. So we will first finish our discussion on this stop and wait protocol.

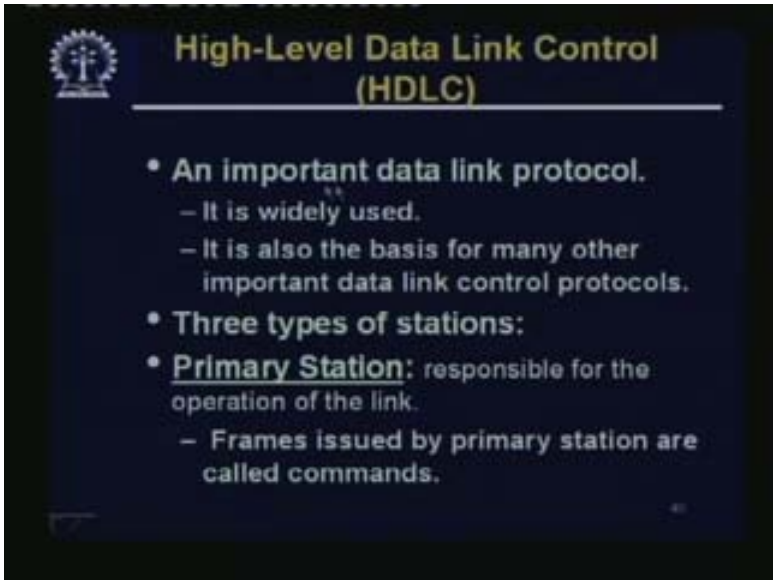
(Refer Slide Time: 54:34)

(Refer Slide Time: 54:39)

And this we have already seen, so each sender station sends one packet which is ACK'ed. If it is comes with some error it may send a negative acknowledgement, it may not arrive at all. In that case the senders will time out in either case that means, it is getting a negative acknowledgement or whether it is getting a time out. It will retransmit that frame. Retransmit whichever frame was just transmitted earlier and the point to note is that there is only one frame, which is in transit in the channel at any point of time. Whether it is the frame or the acknowledgment, now we have discussed some of the major techniques which are used in the data link layer, so what we will do now is that we will look at just one more protocol.

This is somewhat general version that we have discussed. PPP that is the point to point protocol which is used for point to point communication but there could be a point to multi point communication also. For this we have this high level data link control, this is the name of the protocol. So this is the name of the protocol or HDLC in short. I mean this is not so high level any longer this is not considered, so high level any longer but even then this is an important protocol. So there are some variations of this, which are used in various places. So we will just look at some generic the way how this HDLC is important.

(Refer Slide Time: 56:32)



Now this is an important data link protocol this HDLC or some variant say this is widely used, it also is the basis for many other important data link control protocols. So we suggest three types of stations. One could be primary station this is responsible for the operation of the link. By the way you do not have to remember all these because when you are trying to use it or design it for some particular situation you can always look up the reference and see what exactly the different frames are what are their formats etc . I am trying to give you a flavor of the kind of information about which a typical protocol would need to exchange.

So with this example we come to the end of this general data link control, but still we have seen some examples of data link control, for example we have seen some examples of MAC like with token bus, token ring, etc. We will see other examples of MAC as we take up other kinds of networks. Similarly we have seen some kind of error and data control. By the way the similar approach is used in some other higher layers also. We will talk about it when we come to it and we will talk about more specific and more widely used systems in the next class onwards. So in the next class what we will do is, we will look at another type of communication and how MAC is done on that in satellite communication .So that will be the next topic. Thank you.