UNIT – III: Data Link layer

To provide service to the network layer, the data link layer must use the service provided to it by the physical layer. What the physical layer does is accept a raw bit stream and attempt to deliver it to the destination. This bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than the number of bits transmitted, and they may have different values. It is up to the data link layer to detect and, if necessary, correct errors. The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. When a frame arrives at the destination, the checksum is recomputed. If the newly computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it.

Breaking the bit stream up into frames is more difficult than it at first appears. One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text. However, networks rarely make any guarantees about timing, so it is possible these gaps might be squeezed out or other gaps might be inserted during transmission. Since it is too risky to count on timing to mark the start and end of each frame, other methods have been devised. We will look at four methods:

1. Character count.
2. Flag bytes with byte stuffing.
3. Starting and ending flags, with bit stuffing.
4. Physical layer coding violations.

The first framing method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is. This technique is shown in Fig.3.1(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.

www.sakshieducation.com

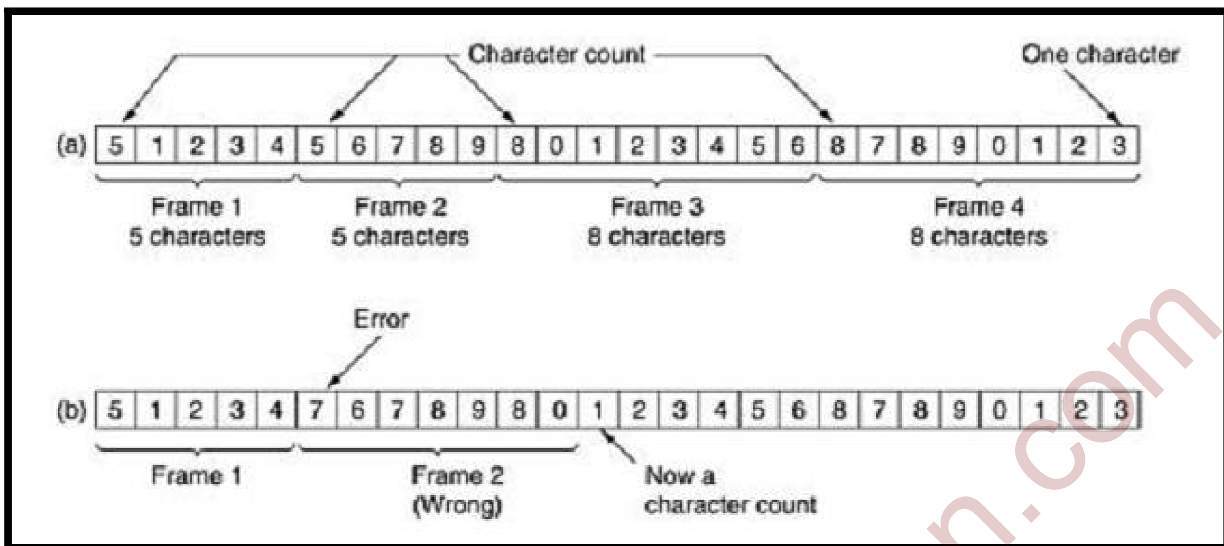


Fig.3.1 A character stream. (a) Without errors. (b) With one error.

The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the character count of 5 in the second frame of Fig. 3.1(b) becomes a 7, the destination will get out of synchronization and will be unable to locate the start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts. Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many characters to skip over to get to the start of the retransmission. For this reason, the character count method is rarely used anymore.

The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes. In the past, the starting and ending bytes were different, but in recent years most protocols have used the same byte, called a flag byte, as both the starting and ending delimiter, as shown in Fig. 3.2(a) as FLAG. In this way, if the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame. Two consecutive flag bytes indicate the end of one frame and start of the next one.

www.sakshieducation.com

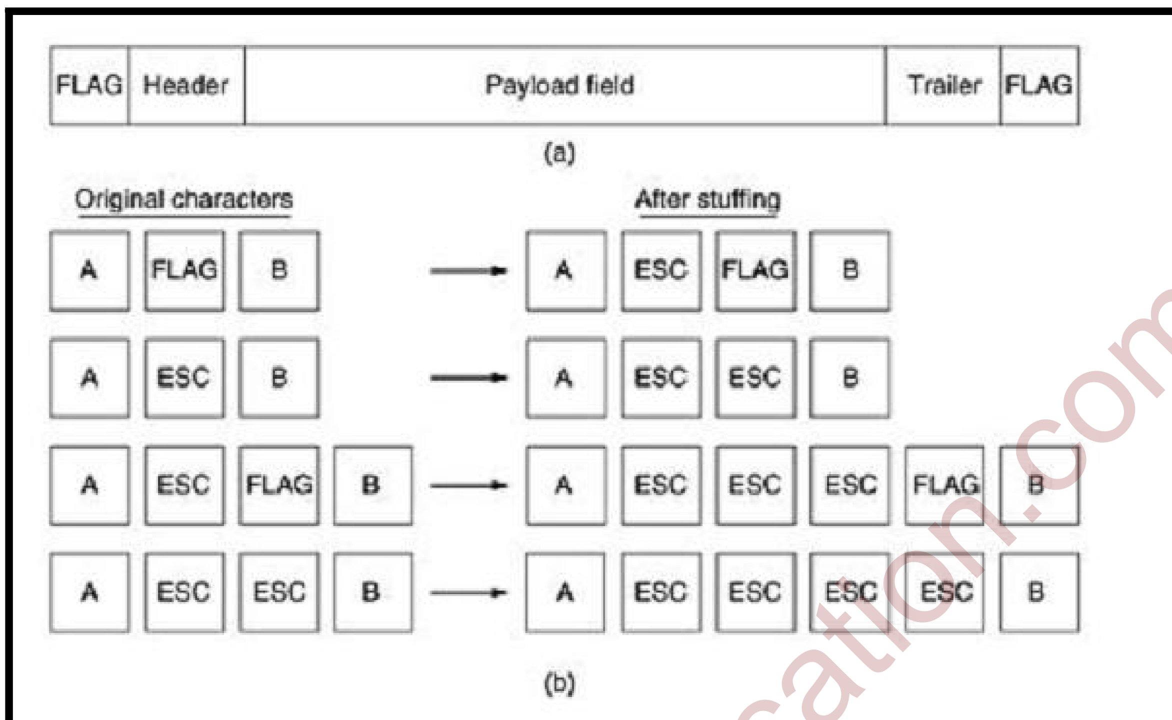


Fig. 3.2 (a) A frame delimited by flag bytes (b) Four examples of byte sequences before and after byte stuffing.

A serious problem occurs with this method when binary data, such as object programs or floating-point numbers, are being transmitted. It may easily happen that the flag byte's bit pattern occurs in the data. This situation will usually interfere with the framing. One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each ''accidental'' flag byte in the data. The data link layer on the receiving end removes the escape byte before the data are given to the network layer. This technique is called byte stuffing or character stuffing. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.

Of course, the next question is: What happens if an escape byte occurs in the middle of the data? The answer is that it, too, is stuffed with an escape byte. Thus, any single escape byte is part of an escape sequence, whereas a doubled one indicates that a single escape occurred

naturally in the data. Some examples are shown in Fig. 3.3(b). In all cases, the byte sequence delivered after de stuffing is exactly the same as the original byte sequence.

The byte-stuffing scheme depicted in Fig. 3.3 is a slight simplification of the one used in the PPP protocol that most home computers use to communicate with their Internet service provider.

A major disadvantage of using this framing method is that it is closely tied to the use of 8-bit characters. Not all character codes use 8-bit characters. For example, UNICODE uses 16-bit characters. As networks developed; the disadvantages of embedding the character code length in the framing mechanism became more and more obvious, so a new technique had to be developed to allow arbitrary sized characters.

The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. It works like this. Each frame begins and ends with a special bit pattern, 01111110 (in fact, a flag byte). Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically de stuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110.

www.sakshieducation.com

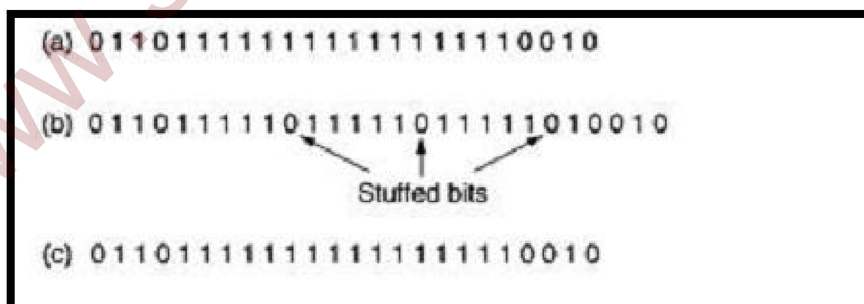


Figure 3.3 Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data. The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy. For example, some LANs encode 1 bit of data by using 2 physical bits. Normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The scheme means that every data bit has a transition in the middle, making it easy for the receiver to locate the bit boundaries. The combinations high-high and low-low are not used for data but are used for delimiting frames in some protocols.

As a final note on framing, many data link protocols use combination of a character count with one of the other methods for extra safety. When a frame arrives, the count field is used to locate the end of the frame. Only if the appropriate delimiter is present at that position and the checksum is correct is the frame accepted as valid. Otherwise, the input stream is scanned for the next delimiter.

3.2 Error correction and detection at the data link layer.

3.2.1 Error-Correcting Codes:

Network designers have developed two basic strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been. The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses error-correcting codes and the latter uses error-detecting codes. The use of error-correcting codes is often referred to as forward error correction.

Each of these techniques occupies a different ecological niche. On channels that are highly reliable, such as fiber, it is cheaper to use an error detecting code and just retransmit the occasional block found to be faulty. However, on channels such as wireless links that make many errors, it is better to add enough redundancy to each block for the receiver to be able to figure out what the original block was, rather than relying on a retransmission, which itself may be in error.

www.sakshieducation.com

To understand how errors can be handled, it is necessary to look closely at what an error really is. Normally, a frame consists of m data (i.e., message) bits and r redundant, or check, bits. Let the total length be n (i.e., n = m + r). An n-bit unit containing data and check bits is often referred to as an n-bit codeword.

Given any two code words, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ. To determine how many bits differ, just exclusive OR the two code words and count the number of 1 bits in the result, for example:

The number of bit positions in which two code words differ is called the Hamming distance. Its significance is that if two codewords are a Hamming distance d apart, it will require d single-bit errors to convert one into the other.

In most data transmission applications, all 2m possible data messages are legal, but due to the way the check bits are computed, not all of the 2n possible codewords are used. Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list find the two codewords whose Hamming distance is minimum. This distance is the Hamming distance of the complete code. The error-detecting and error correcting properties of a code depend on its Hamming distance. To detect d errors, you need a distance d + 1 code because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword. When the receiver sees an invalid codeword, it can tell that a transmission error has occurred. Similarly, to correct d errors, you need a distance 2d +

1 code because that way the legal codewords are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined. As a simple example of an error-detecting code, consider a code in which a single parity b appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100. With odd parity 1011010 becomes 10110101. A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity. It can be used to detect single errors.

As a simple example of an error-correcting code, consider a code with only four valid codewords: 0000000000, 0000011111, 1111100000, and 1111111111

This code has a distance 5, which means that it can correct double errors. If the codeword

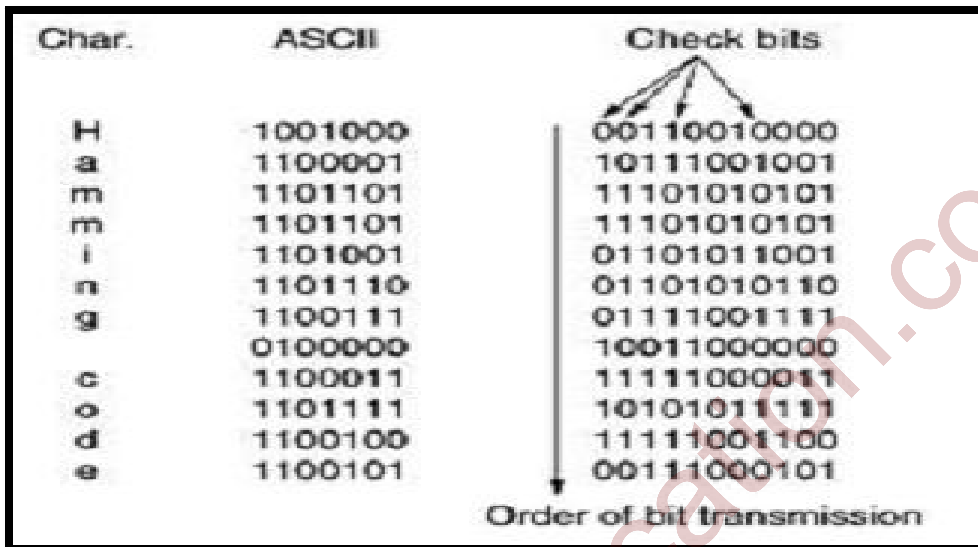
0000000111 arrives, the receiver knows that the original must have been 0000011111. If, however, a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.

Imagine that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected. Each of the 2m legal messages has n illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it. Thus, each of the 2m legal messages requires n + 1 bit patterns dedicated to it. Since the total number of bit patterns is 2n, we must have (n + 1)2m ≤ 2n.

Using n = m + r, this requirement becomes (m + r + 1) ≤ 2r. Given m, this puts a lower limit on the number of check bits needed to correct single errors. This theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950).

The bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2.For example, 11 = 1 + 2 + 8 and 29 = 1 + 4 + 8 + 16. A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8). When a codeword arrives, the receiver initializes a counter to zero. It then examines each check bit, k (k = 1, 2, 4, 8 ...), to see if it has the correct parity. If not, the receiver adds k to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the codeword is accepted as valid. If the counter is nonzero, it contains the number of the incorrect bit. For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only one checked by bits 1, 2, and 8. Figure 4.1

shows some 7-bit ASCII characters encoded as 11-bit code words using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.



Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

Fig.4.1. Use of a Hamming code to correct burst errors

Hamming codes can only correct single errors. However, there is a trick that can be used to permit Hamming codes to correct burst errors. A sequence of k consecutive code words is arranged as a matrix, one codeword per row. Normally, the data would be transmitted one codeword at a time, from left to right. To correct burst errors, the data should be transmitted one column at a time, starting with the leftmost column. When all k bits have been sent, the second column is sent, and so on, as indicated in Fig.4.1. When the frame arrives at the receiver, the matrix is reconstructed, one column at a time. If a burst error of length k occurs, at most 1 bit in each of the k code words will have been affected, but the Hamming code can correct one error per codeword, so the entire block can be restored. This method uses kr check bits to make blocks of km data bits immune to a single burst error of length k or less.

3.2.2 Error-Detecting Codes:

Error-correcting codes are widely used on wireless links, which are notoriously noisy and error prone when compared to copper wire or optical fibers. Without error-correcting

www.sakshieducation.com

codes, it would be hard to get anything through. However, over copper wire or fiber, the error rate is much lower, so error detection and retransmission is usually more efficient there for dealing with the occasional error. As a simple example, consider a channel on which errors are isolated and the error rate is 10-6 per bit. Let the block size be 1000 bits. To provide error correction for 1000-bit blocks, 10 check bits are needed; a megabit of data would require 10,000 check bits. To merely detect a block with a single 1-bit error, one parity bit per block will suffice. Once every 1000 blocks, an extra block (1001 bits) will have to be transmitted. The total overhead for the error detection + retransmission method is only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

If a single parity bit is added to a block and the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5, which is hardly acceptable.

The odds can be improved considerably if each block to be sent is regarded as rectangular matrix n bits wide and k bits high, as described above. A parity bit is computed separately for each column and affixed to the matrix as the last row. The matrix is then transmitted one row at a time. When the block arrives, the receiver checks all the parity bits. If any one of them is wrong, the receiver requests a retransmission of the block. Additional retransmissions are requested as needed until an entire block is received without any parity errors. This method can detect a single burst of length n, since only 1 bit per column will be changed. A burst of length n + 1 will pass undetected, however, if the first bit is inverted, the last bit is inverted, and all the other bits are correct. (A burst error does not imply that all the bits are wrong; it just implies that at least the first and last are wrong.) If the block is badly garbled by a long burst or by multiple shorter bursts, the probability that any of the n columns will have the correct parity, by accident, is 0.5, so the probability of a bad block being accepted when it should not be is 2-n. Although the above scheme may sometimes be adequate, in practice, another method is in widespread use: the polynomial code, also known as a CRC (Cyclic Redundancy Check).

Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from xk-1 to x0. Such a polynomial is said to be of degree k -

www.sakshieducation.com

1. The high order (leftmost) bit is the coefficient of xk-1; the next bit is the coefficient of xk-2, and so on. For example, 110001 has 6 bits and thus represent a six-term polynomial with coefficients 1,

1, 0, 0, 0, and 1: x5 + x4 + x0.

Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. There are no carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR. For example: Long division is carried out the same way as it is in binary except that the subtraction is done modulo 2, as above. A divisor is said ''to go into'' a dividend if the dividend has as many bits as the divisor. When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, G(x), in advance. Both the high- and low-order bits of the generator must be 1. To compute the checksum for some frame with m bits, corresponding to the polynomial M(x), the frame must be longer than the generator polynomial. The idea is to append a checksum to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by G(x). When the receiver gets the checksummed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

The algorithm for computing the checksum is as follows:

1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m + r bits and corresponds to the polynomial xr M(x).

2. Divide the bit string corresponding to G(x) into the bit string corresponding to xr M(x), using modulo 2 division.

3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to xr

M(x) using modulo 2 subtractions. The result is the checksummed frame to be transmitted. Call its polynomial T(x).

Figure illustrates the calculation for a frame 1101011011 using the generator G(x) = x4 + x+ 1.

```
Frame    : 1101011011
Generator:  10011
Message after 4 zero bits are appended:  11010110110000

                 1 1 0 0 0 0 1 0 1 0
        10011 | 1 1 0 1 0 1 1 0 1 1 0 0 0 0
                1 0 0 1 1
                  1 0 0 1 1
                  1 0 0 1 1
                    0 0 0 0 1
                    0 0 0 0 0
                      0 0 0 1 0
                      0 0 0 0 0
                        0 0 1 0 1
                        0 0 0 0 0
                          0 1 0 1 1
                          0 0 0 0 0
                            1 0 1 1 0
                            1 0 0 1 1
                              0 1 0 1 0
                              0 0 0 0 0
                                1 0 1 0 0
                                1 0 0 1 1
                                  0 1 1 1 0
                                  0 0 0 0 0
                                    1 1 1 0   <-- Remainder

Transmitted frame:  11010110111110
```

Fig.5.1. Calculation of the polynomial code checksum

www.sakshieducation.com

3.3 Elementary Data Link Layer Protocols

3.3.1 An Unrestricted Simplex Protocol:

As an initial example we will consider a protocol that is as simple as it can be. Data are transmitted in one direction only. Both the transmitting and receiving network layers are always ready. Processing time can be ignored. Infinite buffer space is available. And best of all, the communication channel between the data link layers never damages or loses frames. This thoroughly unrealistic protocol, which we will nickname ''utopia'' .

The protocol consists of two distinct procedures, a sender and a receiver. The sender runs in the data link layer of the source machine, and the receiver runs in the data link layer of the destination machine. No sequence numbers or acknowledgements are used here, so MAX_SEQ is not needed. The only event type possible is frame_arrival (i.e., the arrival of an undamaged frame).

The sender is in an infinite while loop just pumping data out onto the line as fast as it can. The body of the loop consists of three actions: go fetch a packet from the (always obliging) network layer, construct an outbound frame using the variable s, and send the frame on its way. Only the info field of the frame is used by this protocol, because the other fields have to do with error and flow control and there are no errors or flow control restrictions here. The receiver is equally simple. Initially, it waits for something to happen, the only possibility being the arrival of an undamaged frame. Eventually, the frame arrives and the procedure wait_for_event returns, with event set to frame_arrival (which is ignored anyway). The call to from_physical_layer removes the newly arrived frame from the hardware buffer and puts it in the variable r, where the receiver code can get at it. Finally, the data portion is passed on to the network layer, and the data link layer settles back to wait for the next frame, effectively suspending itself until the frame arrives.

3.3.2 A Simplex Stop-and-Wait Protocol:

The main problem we have to deal with here is how to prevent the sender from flooding the receiver with data faster than the latter is able to process them. In essence, if the receiver requires a time Δt to execute from_physical_layer plus to_network_layer, the

sender must transmit at an average rate less than one frame per time Δt. Moreover, if we assume that no automatic buffering and queuing are done within the receiver's hardware, the sender must never transmit a new frame until the old one has been fetched by from_physical_layer, lest the new one overwrite the old one. In certain restricted circumstances (e.g., synchronous transmission and a receiving data link layer fully dedicated to processing the one input line), it might be possible for the sender to simply insert a delay into protocol 1 to slow it down sufficiently to keep from swamping the receiver. However, more usually, each data link layer will have several lines to attend to, and the time interval between a frame arriving and its being processed may vary considerably. If the network designers can calculate the worst-case behavior of the receiver, they can program the sender to transmit so slowly that even if every frame suffers the maximum delay, there will be no overruns. The trouble with this approach is that it is too conservative. It leads to a bandwidth utilization that is far below the optimum, unless the best and worst cases are almost the same (i.e., the variation in the data link layer's reaction time is small).

A more general solution to this dilemma is to have the receiver provide feedback to the sender. After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame. After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgement) frame arrives. Using feedback from the receiver to let the sender know when it may send more data is an example of the flow control mentioned earlier.

Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait.

3.3.3 A Simplex Protocol for a Noisy Channel:

Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called PAR (Positive Acknowledgement with Retransmission) or ARQ (Automatic Repeat request). Like protocol 2, this one also transmits data only in one direction.

www.sakshieducation.com