ADITYA ENGINEERING COLLEGE (A)

Computer Networks

By

Dr. M. Vamsi Krishna
Professor

Dept of Computer Science and Engineering
Aditya Engineering College(A)
Surampalem.

Unit - II

- **Data link layer**: Design issues, **Framing**: fixed size framing, variable size framing, flow control, error control, error detection and correction codes, CRC, Checksum: idea, one’s complement internet checksum, services provided to Network Layer.
- **Elementary Data Link Layer protocols**: Simplex protocol, Simplex stop and wait, Simplex protocol for Noisy Channel.
- **Sliding window protocol:** One bit, Go back N, Selective repeat, Stop and wait protocol, **Data link layer in HDLC:** configuration and transfer modes, frames, control field, point to point protocol (PPP): framing transition phase, multiplexing, multi link PPP.

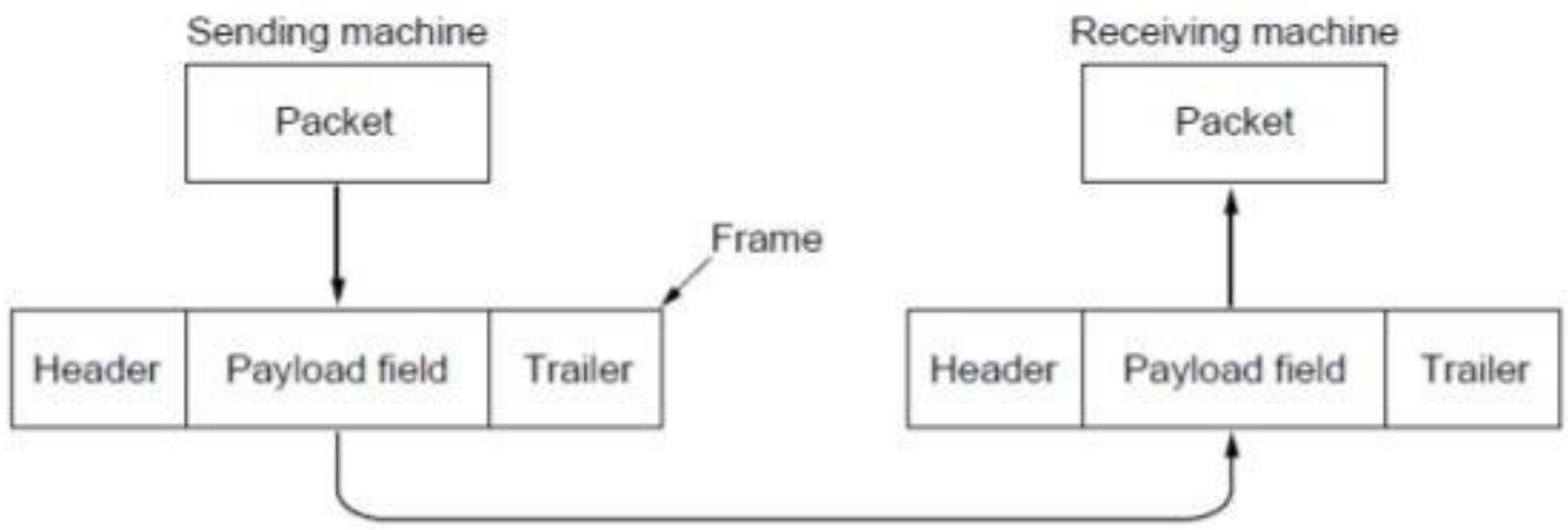
Data link layer design issues

- *The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:*

1. *Providing a well-defined service interface to the network layer.*
2. *Dealing with transmission errors.*
3. *Regulating the flow of data so that slow receivers are not swamped by fast senders.*

- *To accomplish these goals, the data link layer takes the packets it gets from the network layer and encapsulates them into* ***frames*** *for transmission.*
- *Each frame contains a* ***frame header****, a* ***payload field*** *for holding the packet, and a* ***frame trailer****, as illustrated in Fig.*
- *Frame management forms the heart of what the data link layer does.*

Relationship between packets and frames.



Services Provided to the Network Layer

- *The function of the data link layer is to provide services to the network layer.*
- *The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine.*
- *On the source machine is an entity, call it a process, in the network layer that hands some bits to the data link layer for transmission to the destination.*
- *The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there.*
- *The actual transmission follows a path, but it is easier to think in terms of two data link layer processes communicating using a data link protocol.*

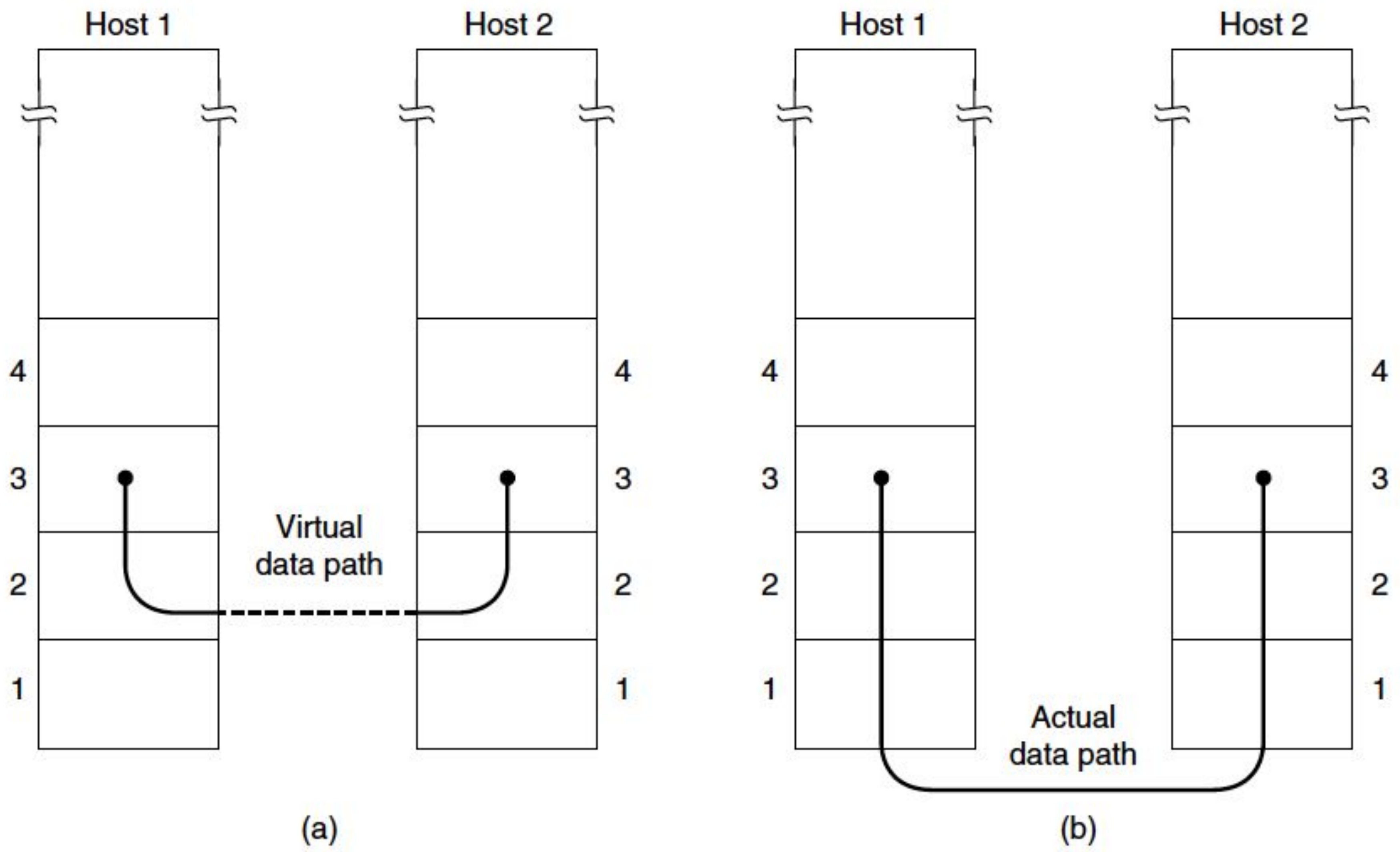


Figure 3-2. (a) Virtual communication. (b) Actual communication.

- *The data link layer can be designed to offer various services.*
- *The actual services that are offered vary from protocol to protocol.*
- *Three reasonable possibilities that are considered are:*

1. *Unacknowledged connectionless service.*
2. *Acknowledged connectionless service.*
3. *Acknowledged connection-oriented service.*

Unacknowledged connectionless

- *Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.*
- *Ethernet is a good example of a data link layer that provides this class of service.*
- *This class of service is appropriate when the error rate is very low, so recovery is left to higher layers.*

Acknowledged connectionless

- *When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged.*
- *In this way, the sender knows whether a frame has arrived correctly or been lost.*
- *If it has not arrived within a specified time interval, it can be sent again.*
- *This service is useful over unreliable channels, such as wireless systems.*
- *802.11 (WiFi) is a good example of this class of service.*

Acknowledged connection-oriented

- *the most sophisticated service the data link layer can provide to the network layer is connection-oriented service.*
- *With this service, the source and destination machines establish a connection before any data are transferred.*
- *Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received.*
- *Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order.*
- *Connection-oriented service thus provides the network layer processes with the equivalent of a reliable bit stream.*
- *It is appropriate over long, unreliable links such as a satellite channel or a long-distance telephone circuit.*
- *If acknowledged connectionless service were used, it is conceivable that lost acknowledgements could cause a frame to be sent and received several times, wasting bandwidth.*

- *When connection-oriented service is used, transfers go through three distinct phases.*
- *In the first phase, the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not.*
- *In the second phase, one or more frames are actually transmitted.*
- *In the third and final phase, the connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.*

Framing

- *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.*
- *If the channel is noisy, as it is for most wireless and some wired links, the physical layer will add some redundancy to its signals to reduce the bit error rate to a tolerable level.*
- *However, the bit stream received by the data link layer is not guaranteed to be error free.*
- *Some bits may have different values and the number of bits received may be less than, equal to, or more than the number of bits transmitted.*
- *It is up to the data link layer to detect and, if necessary, correct errors.*

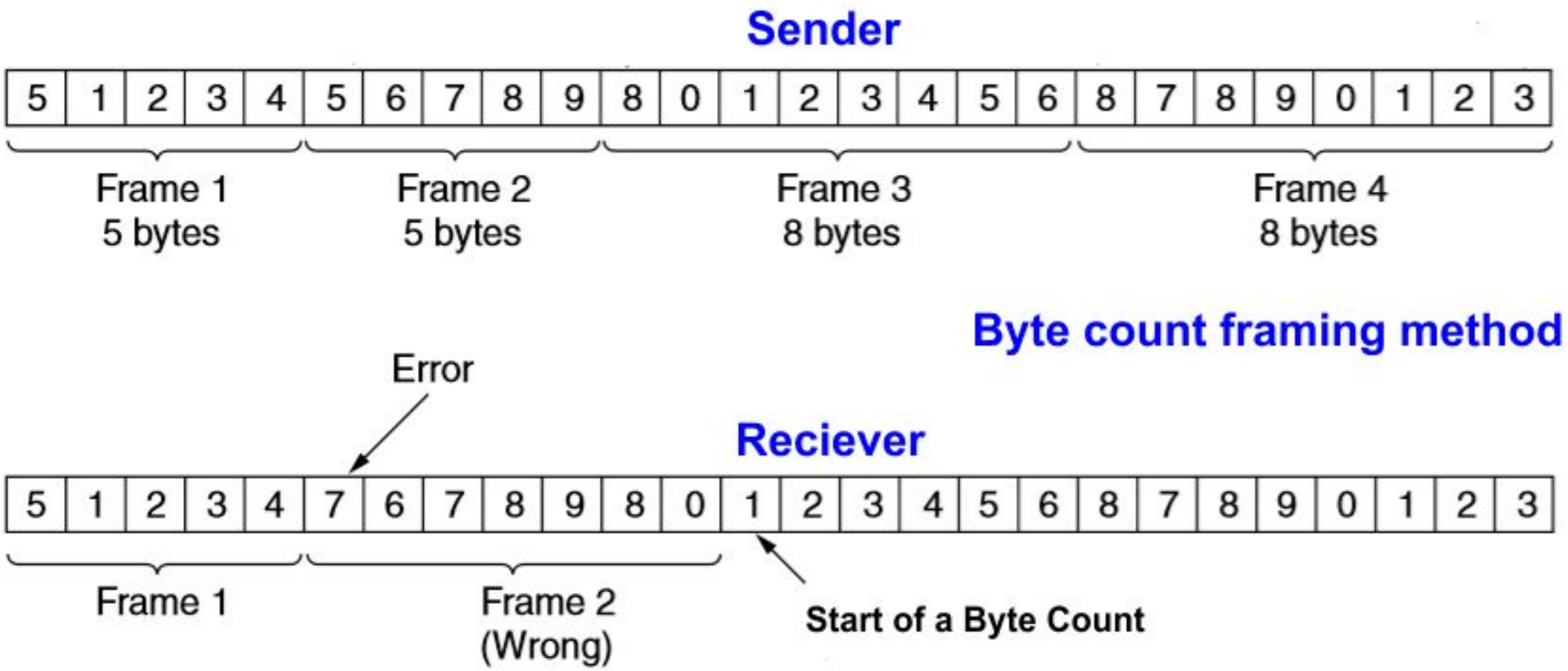
- *The usual approach is for the data link layer to break up the bit stream into discrete frames, compute a short token called a checksum for each frame, and include the checksum in the frame when it is transmitted.*
- *When a frame arrives at the destination, the checksum is recomputed. If the newly 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.*

- *A good design must make it easy for a receiver to find the start of new frames while using little of the channel bandwidth.*
- *We will look at four methods:*

1. *Byte count.*
2. *Flag bytes with byte stuffing.*
3. *Flag bits with bit stuffing.*
4. *Physical layer coding violations*

Byte count

- *The first framing method uses a field in the header to specify the number of bytes in the frame.*
- *When the data link layer at the destination sees the byte count, it knows how many bytes follow and hence where the end of the frame is.*
- *This technique is shown in Figure for four small example frames of sizes 5, 5, 8, and 8 bytes, respectively.*


Sender
5 1 2 3 4 5 6 7 8 9 8 0 1 2 3 4 5 6 8 7 8 9 0 1 2 3
Frame 1 5 bytes
Frame 2 5 bytes
Frame 3 8 bytes
Frame 4 8 bytes
Byte count framing method
Error
Reciever
5 1 2 3 4 7 6 7 8 9 8 0 1 2 3 4 5 6 8 7 8 9 0 1 2 3
Frame 1
Frame 2 (Wrong)
Start of a Byte Count

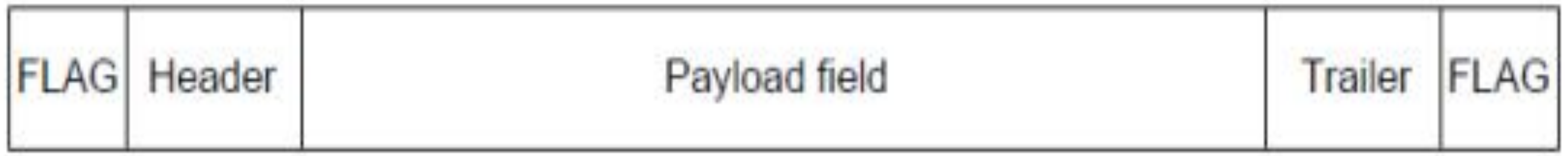
Flag bytes with byte stuffing

- *The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes.*
- *Often the same byte, called a **flag byte**, is used as both the starting and ending delimiter.*
- *Two consecutive flag bytes indicate the end of one frame and the start of the next.*
- *Thus, if the receiver ever loses synchronization it can just search for two flag bytes to find the end of the current frame and the start of the next frame.*

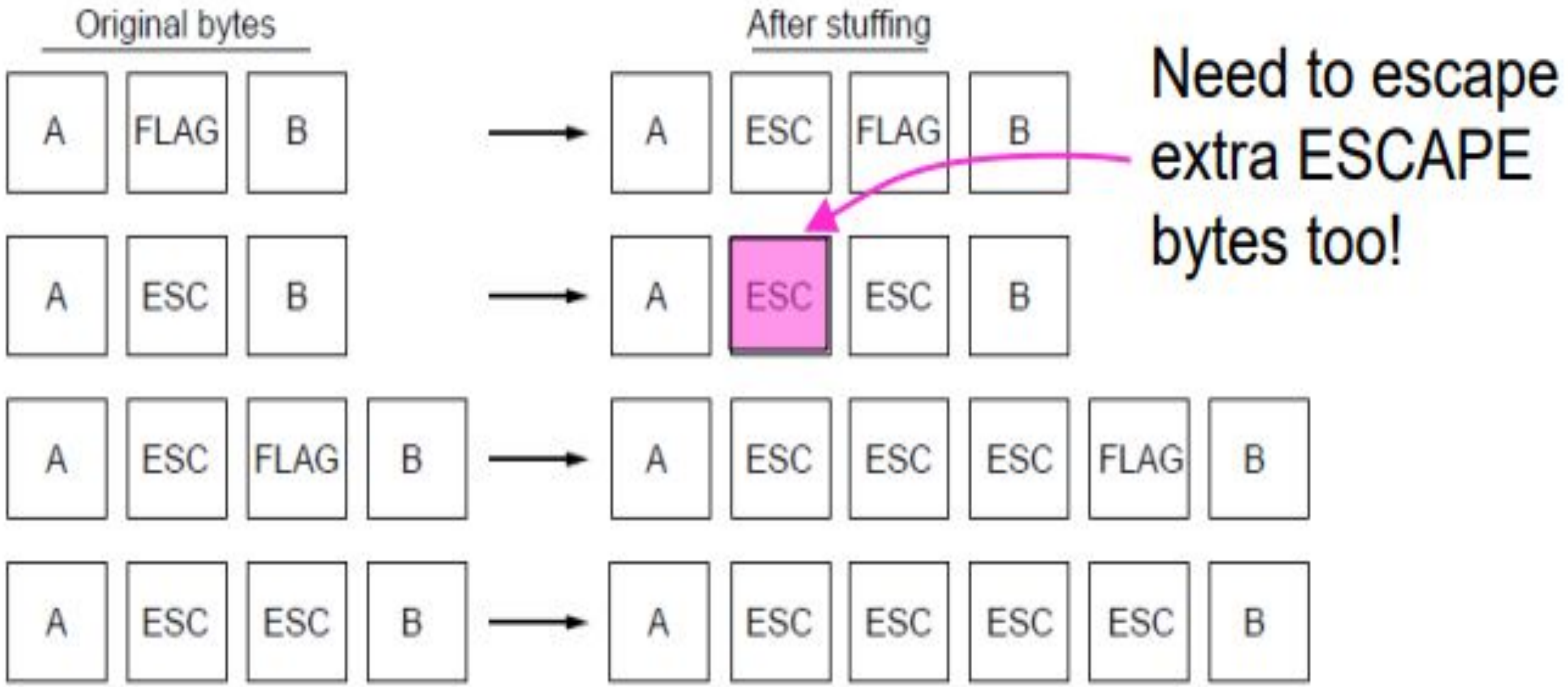
- *It may happen that the flag byte occurs in the data, especially when binary data such as photographs or songs are being transmitted.*
- *This situation would interfere with the framing. One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each ''accidental'' flag byte in the data.*
- *Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.*
- *The data link layer on the receiving end removes the escape bytes before giving the data to the network layer.*
- *This technique is called byte stuffing.*

- *What happens if an escape byte occurs in the middle of the data?*
- *it, too, is stuffed with an escape byte.*
- *At the receiver, the first escape byte is removed, leaving the data byte that follows it*
- *Some examples are shown in Figure. In all cases, the byte sequence delivered after destuffing is exactly the same as the original byte sequence.*
- *We can still search for a frame boundary by looking for two flag bytes in a row, without bothering to undo escapes.*

Frame format



Stuffing examples


Original bytes
After stuffing
A FLAG B → A ESC FLAG B
A ESC B → A ESC ESC B
A ESC FLAG B → A ESC ESC ESC FLAG B
A ESC ESC B → A ESC ESC ESC ESC B
Need to escape extra ESCAPE bytes too!

Flag bits with bit stuffing.

- *The third method of delimiting the bit stream gets around a disadvantage of byte stuffing, which is that it is tied to the use of 8-bit bytes.*
- *Framing can be also be done at the bit level, so frames can contain an arbitrary number of bits made up of units of any size.*
- *Each frame begins and ends with a special bit pattern, 01111110 or 0x7E in hexadecimal.*
- *This pattern is 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.*

- *When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (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.*
- *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.*
- *Figure gives an example of bit stuffing.*

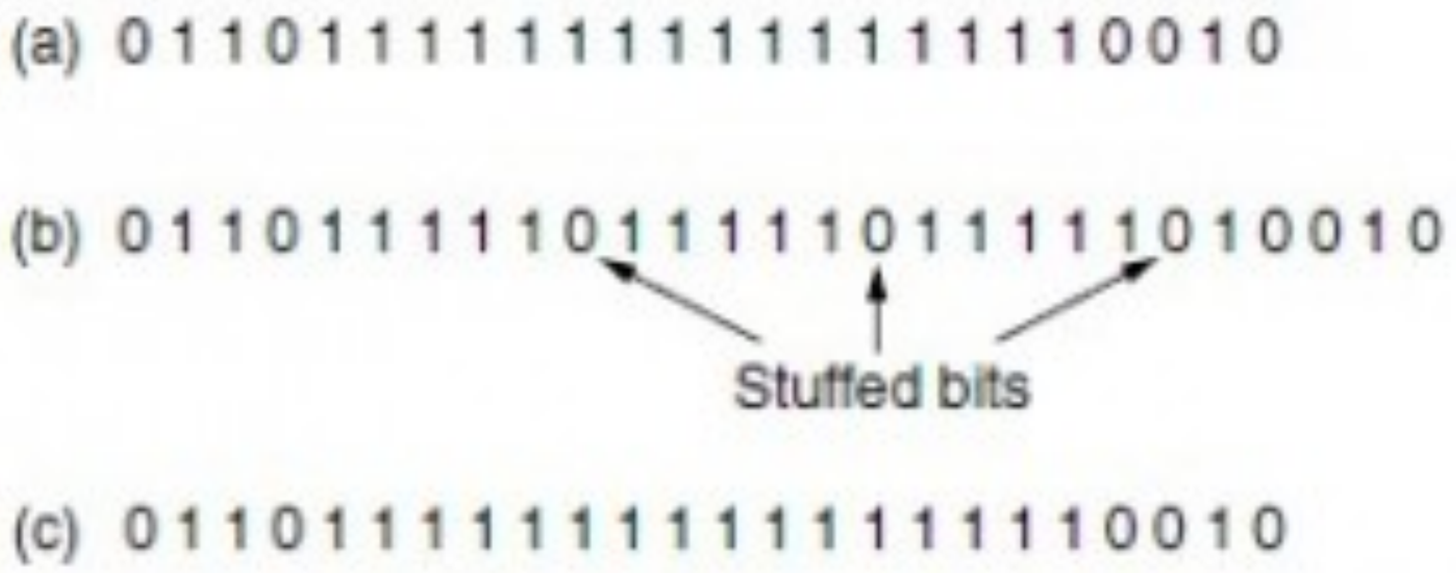

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0
Stuffed bits
(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Figure 3-5. 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.

Physical layer coding violations

- *The last method of framing is to use a shortcut from the physical layer.*
- *the encoding of bits as signals often includes redundancy to help the receiver.*
- *This redundancy means that some signals will not occur in regular data.*
- *For example, in the 4B/5B line code 4 data bits are mapped to 5 signal bits to ensure sufficient bit transitions. This means that 16 out of the 32 signal possibilities are not used.*
- *We can use some reserved signals to indicate the start and end of frames.*
- *In effect, we are using ''coding violations'' to delimit frames.*
- *The beauty of this scheme is that, because they are reserved signals, it is easy to find the start and end of frames and there is no need to stuff the data.*

- *Many data link protocols use a combination of these methods for safety.*
- *A common pattern used for Ethernet and 802.11 is to have a frame begin with a well-defined pattern called a* ***preamble****.*
- *This pattern might be quite long (72 bits is typical for 802.11) to allow the receiver to prepare for an incoming packet.*
- *The preamble is then followed by a length (i.e., count) field in the header that is used to locate the end of the frame.*

Error Control

- *Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination and in the proper order.*
- *For unacknowledged connectionless service it might be fine if the sender just kept outputting frames without regard to whether they were arriving properly. But for reliable, connection-oriented service it would not be fine at all.*

- *The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line.*
- *Typically, the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames.*
- *If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely.*
- *On the other hand, a negative acknowledgement means that something has gone wrong and the frame must be transmitted again.*

- *An additional complication comes from the possibility that hardware troubles may cause a frame to vanish completely. In this case:*
- *The receiver will not react at all, since it has no reason to react.*
- *Similarly, if the acknowledgement frame is lost, the sender will not know how to proceed.*
- *It should be clear that a protocol in which the sender transmits a frame and then waits for an acknowledgement, positive or negative, will hang forever if a frame is ever lost.*

- *This possibility is dealt with by introducing timers into the data link layer.*
- *When the sender transmits a frame, it generally also starts a timer.*
- *The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender.*
- *Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out, in which case the timer will be cancelled.*

- *However, if either the frame or the acknowledgement is lost, the timer will go off, alerting the sender to a potential problem.*
- *The obvious solution is to just transmit the frame again. However, when frames may be transmitted multiple times there is a danger that the receiver will accept the same frame two or more times and pass it to the network layer more than once.*
- *To prevent this from happening, it is generally necessary to assign sequence numbers to outgoing frames, so that the receiver can distinguish retransmissions from originals.*

Flow Control

- *Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them.*
- *This situation can occur when the sender is running on a fast, powerful computer and the receiver is running on a slow, low-end machine.*

- *Two approaches are commonly used.*
- *In the first one,* ***feedback-based flow control****, the receiver sends back information to the sender giving it permission to send more data, or at least telling the sender how the receiver is doing.*
- *In the second one,* ***rate-based flow control****, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.*

Error Detection and Correction

- *Network designers have developed two basic strategies for dealing with errors.*
- *Both add redundant information to the data that is sent.*
- *One strategy is to include enough redundant information to enable the receiver to deduce what the transmitted data must have been.*
- *The other is to include only enough redundancy to allow the receiver to deduce that an error has 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 FEC (Forward Error Correction).*

Error-Correcting Codes

- *We have four different error-correcting codes:*

1. *Hamming codes.*
2. *Binary convolutional codes.*
3. *Reed-Solomon codes.*
4. *Low-Density Parity Check codes.*

- *All of these codes add redundancy to the information that is sent.*
- *A frame consists of m data (i.e., message) bits and r redundant (i.e. check) bits.*

- *In a **block code**, the r check bits are computed solely as a function of the m data bits with which they are associated, as though the m bits were looked up in a large table to find their corresponding r check bits.*
- *In a **systematic code**, the m data bits are sent directly, along with the check bits, rather than being encoded themselves before they are sent.*
- *In a **linear code**, the r check bits are computed as a linear function of the m data bits. Exclusive OR (XOR) or modulo 2 addition is a popular choice.*
- *This means that encoding can be done with operations such as matrix multiplications or simple logic circuits.*

Hamming codes

- *Given any two codewords that may be transmitted or received—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 XOR the two codewords and count the number of 1 bits in the result.*

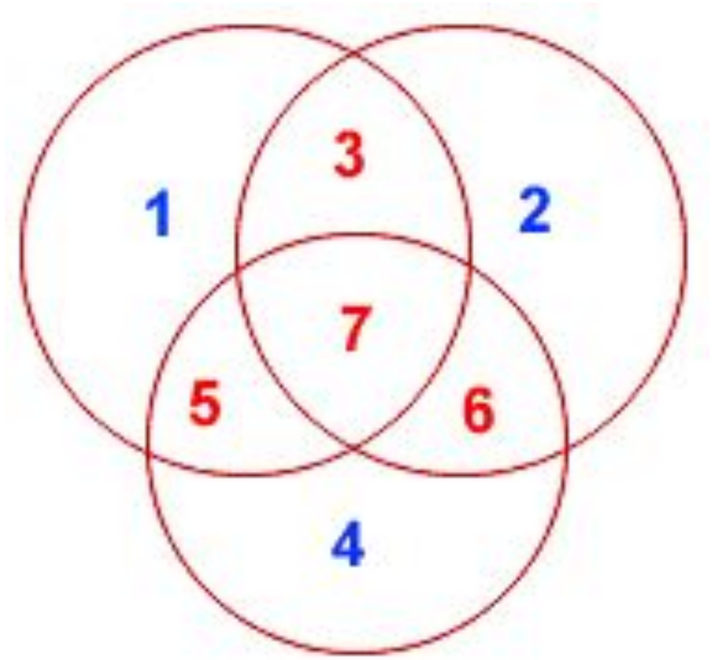
$$\begin{array}{c} 10001001 \\ \underline{10110001} \\ 00111000 \end{array}$$

- *The number of bit positions in which two codewords differ is called the* ***Hamming distance*** *(Hamming, 1950).*
- *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.*

- Consider a message having four data bits (D) which is to be transmitted as a 7-bit codeword by adding three error control bits. This would be called a (7,4) code. The three bits to be added are three EVEN Parity bits (P), where the parity of each is computed on different subsets of the message bits as shown below.

7	6	5	4	3	2	1	
D	D	D	P	D	P	P	7-BIT CODEWORD
D	-	D	-	D	-	P	(EVEN PARITY)
D	D	-	-	D	P	-	(EVEN PARITY)
D	D	D	P	-	-	-	(EVEN PARITY)

- **Why Those Bits?** - The three parity bits **(1,2,4)** are related to the data bits **(3,5,6,7)** as shown at right. In this diagram, each overlapping circle corresponds to one parity bit and defines the four bits contributing to that parity computation. For example, data bit **3** contributes to parity bits **1** and **2**. Each circle (parity bit) encompasses a total of four bits, and each circle must have EVEN parity. Given four data bits, the three parity bits can easily be chosen to ensure this condition.
- It can be observed that changing any one bit numbered 1..7 uniquely affects the three parity bits. Changing bit **7** affects all three parity bits, while an error in bit **6** affects only parity bits **2** and **4**, and an error in a parity bit affects only that bit. The location of any single bit error is determined directly upon checking the three parity circles.

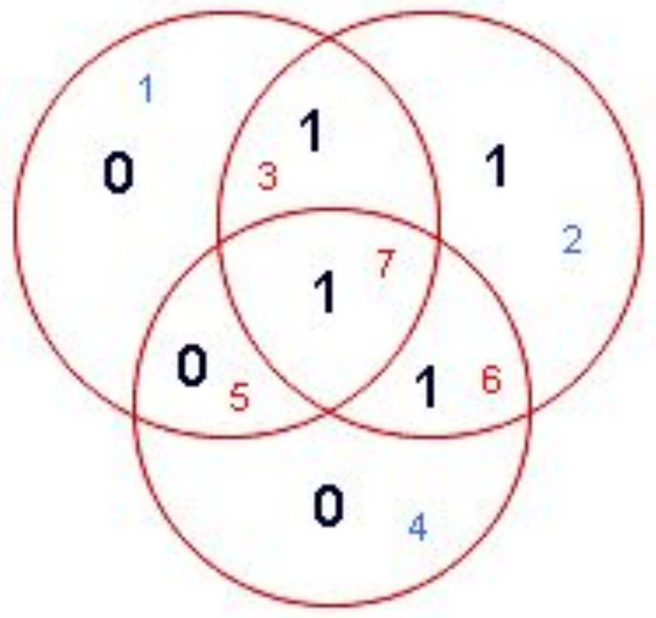

3
1
2
7
5
6
4

- For example, the message 1101 would be sent as 1100110, since:

7	6	5	4	3	2	1	
1	1	0	0	1	1	0	7-BIT CODEWORD
1	-	0	-	1	-	**0**	(EVEN PARITY)
1	1	-	-	1	**1**	-	(EVEN PARITY)
1	1	0	**0**	-	-	-	(EVEN PARITY)

- When these seven bits are entered into the parity circles, it can be confirmed that the choice of these three parity bits ensures that the parity within each circle is EVEN, as shown here.

7	6	5	4	3	2	1			
1	1	1	0	1	1	0	7-BIT CODEWORD		
1	-	1	-	1	-	0	(EVEN PARITY)	NOT!	1
1	1	-	-	1	1	-	(EVEN PARITY)	OK!	0
1	1	1	0	-	-	-	(EVEN PARITY)	NOT!	1



Error-Detecting Codes

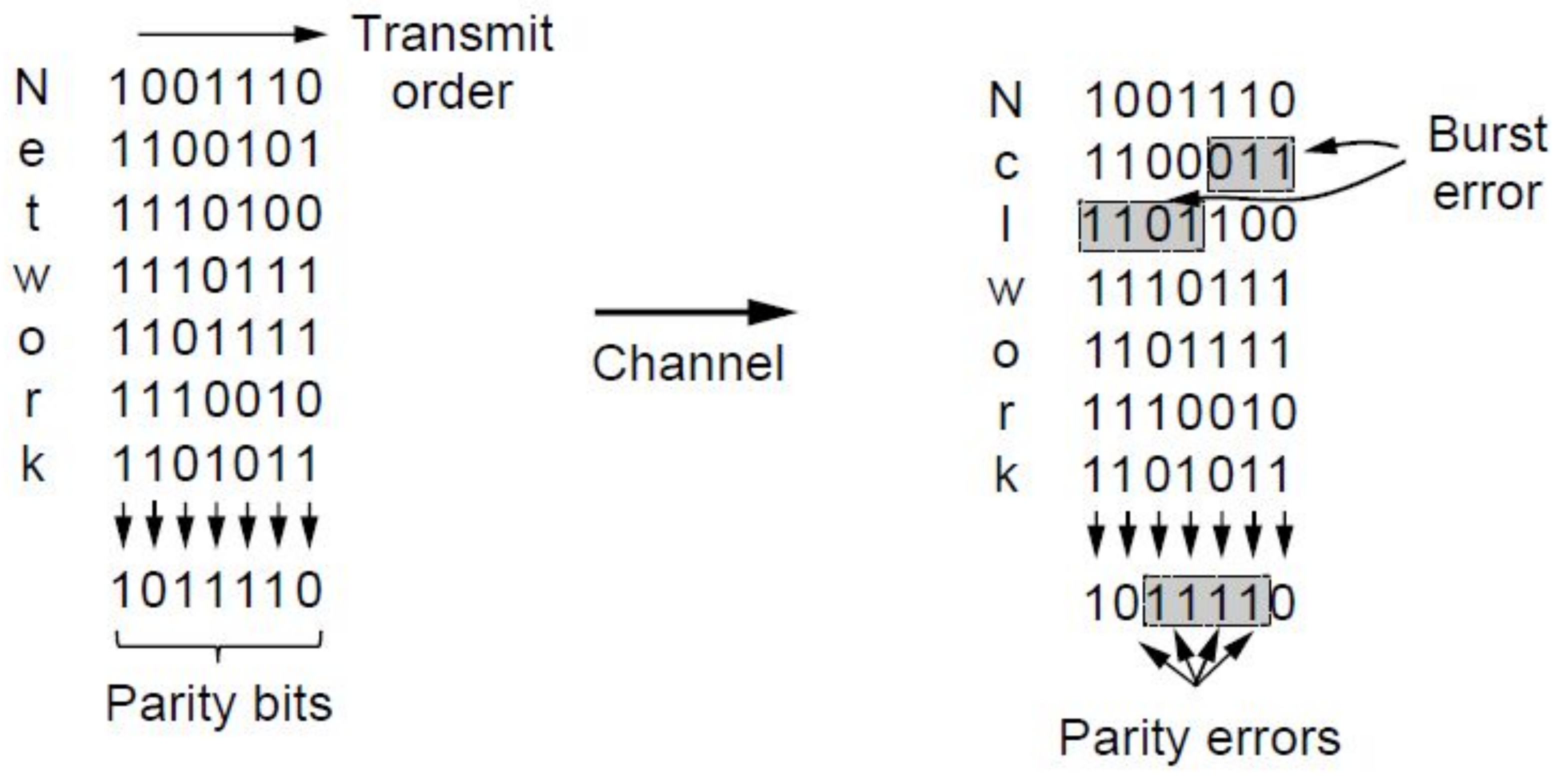
Linear, systematic block codes

1. Parity.
2. Checksums.
3. Cyclic Redundancy Checks (CRCs).

- *Error – detecting codes are more efficient than error-correcting codes.*
- *Consider the first error-detecting code, in which a single parity bit is appended to the data.*
- *The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd).*
- *Doing this is equivalent to computing the (even) parity bit as the modulo 2 sum or XOR of the data bits.*
- *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 of 2, since any single-bit error produces a codeword with the wrong parity.*
- *This means that it can detect single-bit errors.*

- *One difficulty with this scheme is that a single parity bit can only reliably detect a single-bit error in the block.*
- *If 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 a rectangular matrix n bits wide and k bits high.*
- *Now, if we compute and send one parity bit for each row, up to k bit errors will be reliably detected as long as there is at most one error per row.*

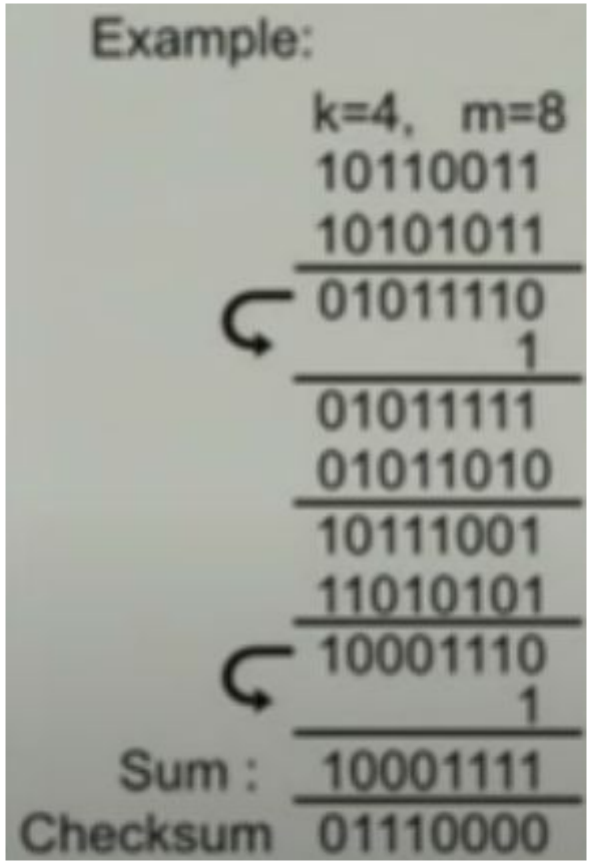
Parity Bits


Transmit order
N 1001110
e 1100101
t 1110100
w 1110111
o 1101111
r 1110010
k 1101011
1011110
Parity bits
Channel
N 1001110
c 1100011
l 1101100
w 1110111
o 1101111
r 1110010
k 1101011
1011110
Burst error
Parity errors

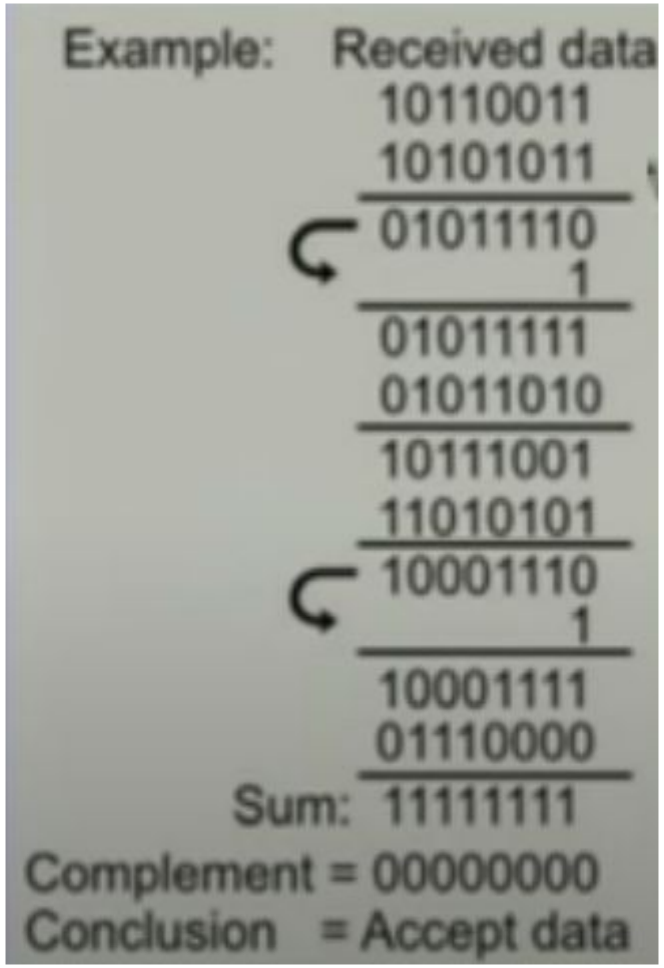
- *The second kind of error-detecting code, the checksum, is closely related to groups of parity bits.*
- *The word ‘‘checksum’’ is often used to mean a group of check bits associated with a message, regardless of how are calculated.*
- *A group of parity bits is one example of a checksum.*
- *However, there are other, stronger checksums based on a running sum of the data bits of the message.*
- *The checksum is usually placed at the end of the message, as the complement of the sum function.*
- *This way, errors may be detected by summing the entire received codeword, both data bits and checksum.*
- *If the result comes out to be zero, no error has been detected.*

10110011 10101011 01011010 11010101

Checksum=**01110000**



```
Example:
                 k=4,   m=8
                 10110011
                 10101011
                 --------
                 01011110
                        1
                 --------
                 01011111
                 01011010
                 --------
                 10111001
                 11010101
                 --------
                 10001110
                        1
                 --------
       Sum :     10001111
                 --------
   Checksum      01110000
```



```
Example:     Received data
                 10110011
                 10101011
                 --------
                 01011110
                        1
                 --------
                 01011111
                 01011010
                 --------
                 10111001
                 11010101
                 --------
                 10001110
                        1
                 --------
                 10001111
                 01110000
                 --------
           Sum:  11111111
Complement = 00000000
Conclusion   = Accept data
```

CRC

- *Although the two preceding schemes may sometimes be adequate at higher layers, in practice, a third and stronger kind of error-detecting code is in widespread use at the link layer: the CRC (Cyclic Redundancy Check), also known as a polynomial code.*
- *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 x^{k-1} to x^{0}.*
- *Such a polynomial is said to be of degree $k - 1$. The high-order (leftmost) bit is the coefficient of x^{k-1}, the next bit is the coefficient of x^{k-2}, and so on.*

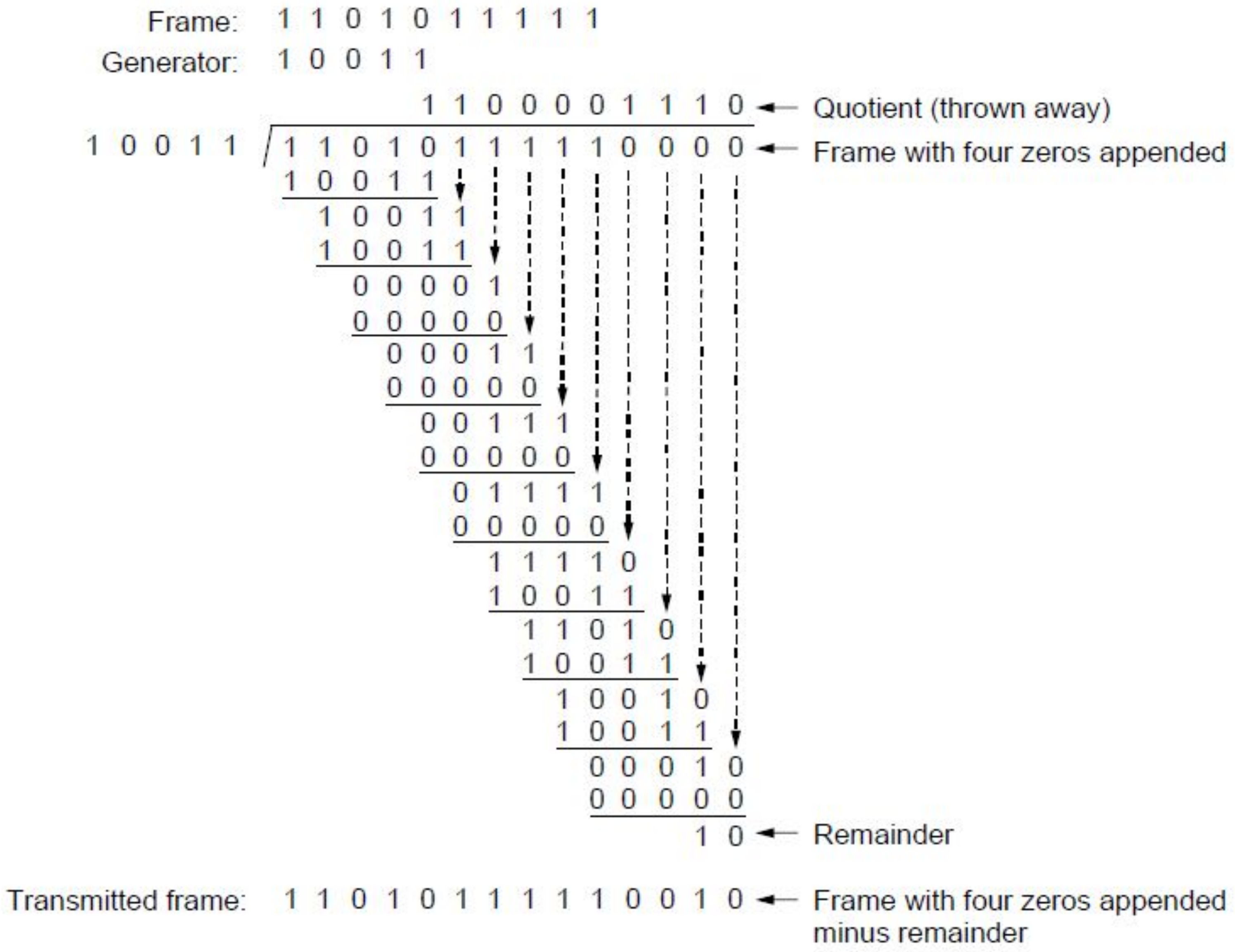
- *For example, 110001 has 6 bits and thus represents a six-term polynomial with*
- *coefficients 1, 1, 0, 0, 0, and 1:* $1x^5 + 1x^4 + 0x^3 + 0x^2 + 0x^1 + 1x^0$.
- *Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory.*
- *It does not have carries for addition or borrows for subtraction.*
- *Both addition and subtraction are identical to exclusive OR.*

$$\begin{array}{r} 10011011 \\ +\ 11001010 \\ \hline 01010001 \end{array} \qquad \begin{array}{r} 00110011 \\ +\ 11001101 \\ \hline 11111110 \end{array} \qquad \begin{array}{r} 11110000 \\ -\ 10100110 \\ \hline 01010110 \end{array} \qquad \begin{array}{r} 01010101 \\ -\ 10101111 \\ \hline 11111010 \end{array}$$

- *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 loworder bits of the generator must be 1. To compute the CRC 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 CRC 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 CRC is as follows:

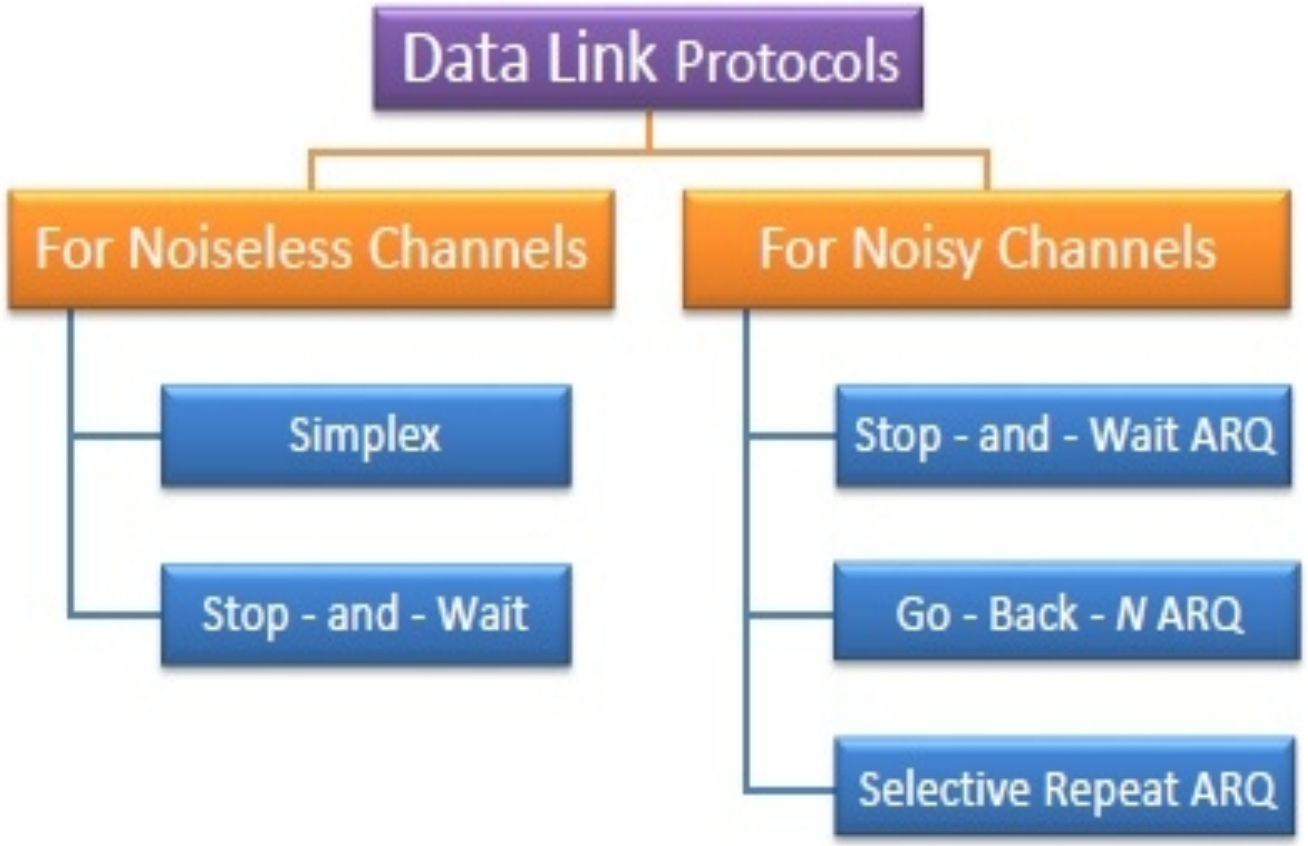
1. *Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m + r bits and corresponds to the polynomial x rM(x).*
2. *Divide the bit string corresponding to G(x) into the bit string corresponding to x rM(x), using modulo 2 division.*
3. *Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to x rM(x) using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).*



```
            Frame:  1 1 0 1 0 1 1 1 1 1
        Generator:  1 0 0 1 1

                             1 1 0 0 0 0 1 1 1 0   ← Quotient (thrown away)
          1 0 0 1 1 / 1 1 0 1 0 1 1 1 1 1 0 0 0 0   ← Frame with four zeros appended
                      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 1
                            0 0 0 0 0
                              0 0 1 1 1
                              0 0 0 0 0
                                0 1 1 1 1
                                0 0 0 0 0
                                  1 1 1 1 0
                                  1 0 0 1 1
                                    1 1 0 1 0
                                    1 0 0 1 1
                                      1 0 0 1 0
                                      1 0 0 1 1
                                        0 0 0 1 0
                                        0 0 0 0 0
                                              1 0   ← Remainder

Transmitted frame:  1 1 0 1 0 1 1 1 1 1 0 0 1 0   ← Frame with four zeros appended
                                                      minus remainder
```

Elementary Data Link Layer protocols

- Simplex protocol
- Simplex stop and wait
- Simplex protocol for Noisy Channel.


Data Link Protocols
For Noiseless Channels
For Noisy Channels
Simplex
Stop - and - Wait
Stop - and - Wait ARQ
Go - Back - N ARQ
Selective Repeat ARQ

A Utopian Simplex Protocol

- 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.
- The communication channel between the data link layers never damages or loses frames.
- The Protocol has distinct procedures for sender and receiver.
- It is hypothetical since it does not handle flow control or error control.

- This is an unrealistic protocol, that’s why it is known as **Utopian Simplex Protocol.**
- No sequence numbers or acknowledgement are used here.

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
 frame s;                             /* buffer for an outbound frame */
 packet buffer;                       /* buffer for an outbound packet */

 while (true) {
      from_network_layer(&buffer);    /* go get something to send */
      s.info = buffer;                /* copy it into s for transmission */
      to_physical_layer(&s);          /* send it on its way */
 }                                    /* Tomorrow, and tomorrow, and tomorrow,
                                         Creeps in this petty pace from day to day
                                         To the last syllable of recorded time.
                                             – Macbeth, V, v */
}

void receiver1(void)
{
 frame r;
 event_type event;                    /* filled in by wait, but not used here */

 while (true) {
      wait_for_event(&event);         /* only possibility is frame_arrival */
      from_physical_layer(&r);        /* go get the inbound frame */
      to_network_layer(&r.info);      /* pass the data to the network layer */
 }
}
```

Figure 3-12. A utopian simplex protocol.

- *The utopia protocol is unrealistic because it does not handle either flow control or error correction.*
- *Its processing is close to that of an unacknowledged connectionless service that relies on higher layers to solve these problems, though even an unacknowledged connectionless service would do some error detection.*

A Simplex Stop-and-Wait Protocol for an Error-Free Channel

- *To tackle the problem of preventing the sender from flooding the receiver with frames faster than the latter is able to process them.*
- *This situation can easily happen in practice so being able to prevent it is of great importance.*
- *The communication channel is still assumed to be error free, however, and the data traffic is still simplex.*

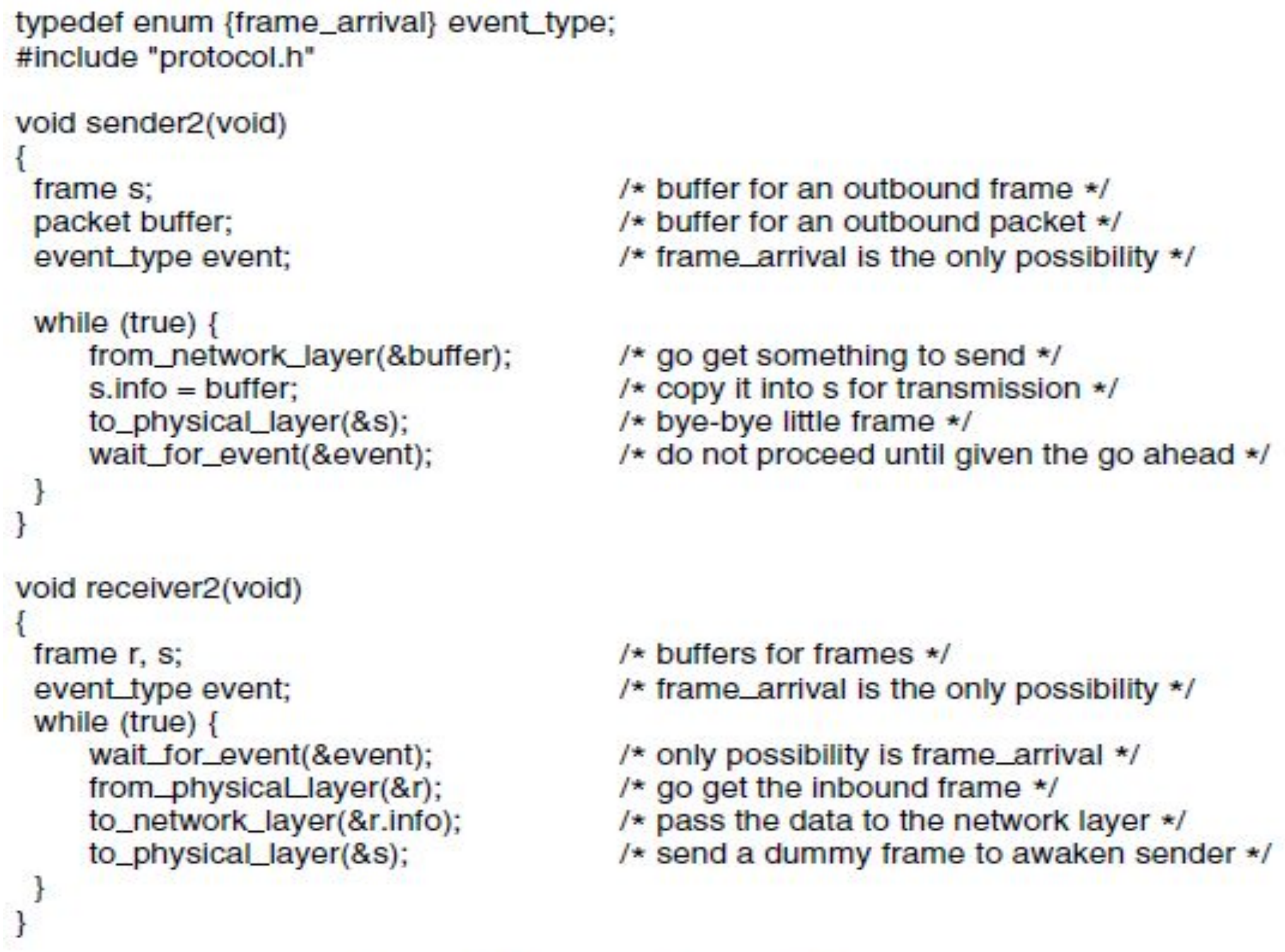
- *One solution is to build the receiver to be powerful enough to process a continuous stream of back-to-back frames.*
- *It must have sufficient buffering and processing abilities to run at the line rate and must be able to pass the frames that are received to the network layer quickly enough.*
- *However, this is a worst-case solution. It requires dedicated hardware and can be wasteful of resources if the utilization of the link is mostly low.*
- *Moreover, it just shifts the problem of dealing with a sender that is too fast elsewhere; in this case to the network layer.*

- *A more general solution to this problem 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.*
- *This delay is a simple example of a flow control protocol.*
- *Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait.*

- *Although data traffic in this example is simplex, going only from the sender to the receiver, frames do travel in both directions.*
- *Consequently, the communication channel between the two data link layers needs to be capable of bidirectional information transfer.*
- *However, this protocol entails a strict alternation of flow: first the sender sends a frame, then the receiver sends a frame, then the sender sends another frame, then the receiver sends another one, and so on.*
- *A half duplex physical channel would suffice here.*

- *As in protocol 1, the sender starts out by fetching a packet from the network layer, using it to construct a frame, and sending it on its way.*
- *But now, unlike in protocol 1, the sender must wait until an acknowledgement frame arrives before looping back and fetching the next packet from the network layer.*
- *The sending data link layer need not even inspect the incoming frame as there is only one possibility.*
- *The incoming frame is always an acknowledgement.*

- *The only difference between receiver1 and receiver2 is that after delivering a packet to the network layer, receiver2 sends an acknowledgement frame back to the sender before entering the wait loop again.*
- *Because only the arrival of the frame back at the sender is important, not its contents, the receiver need not put any particular information in it.*



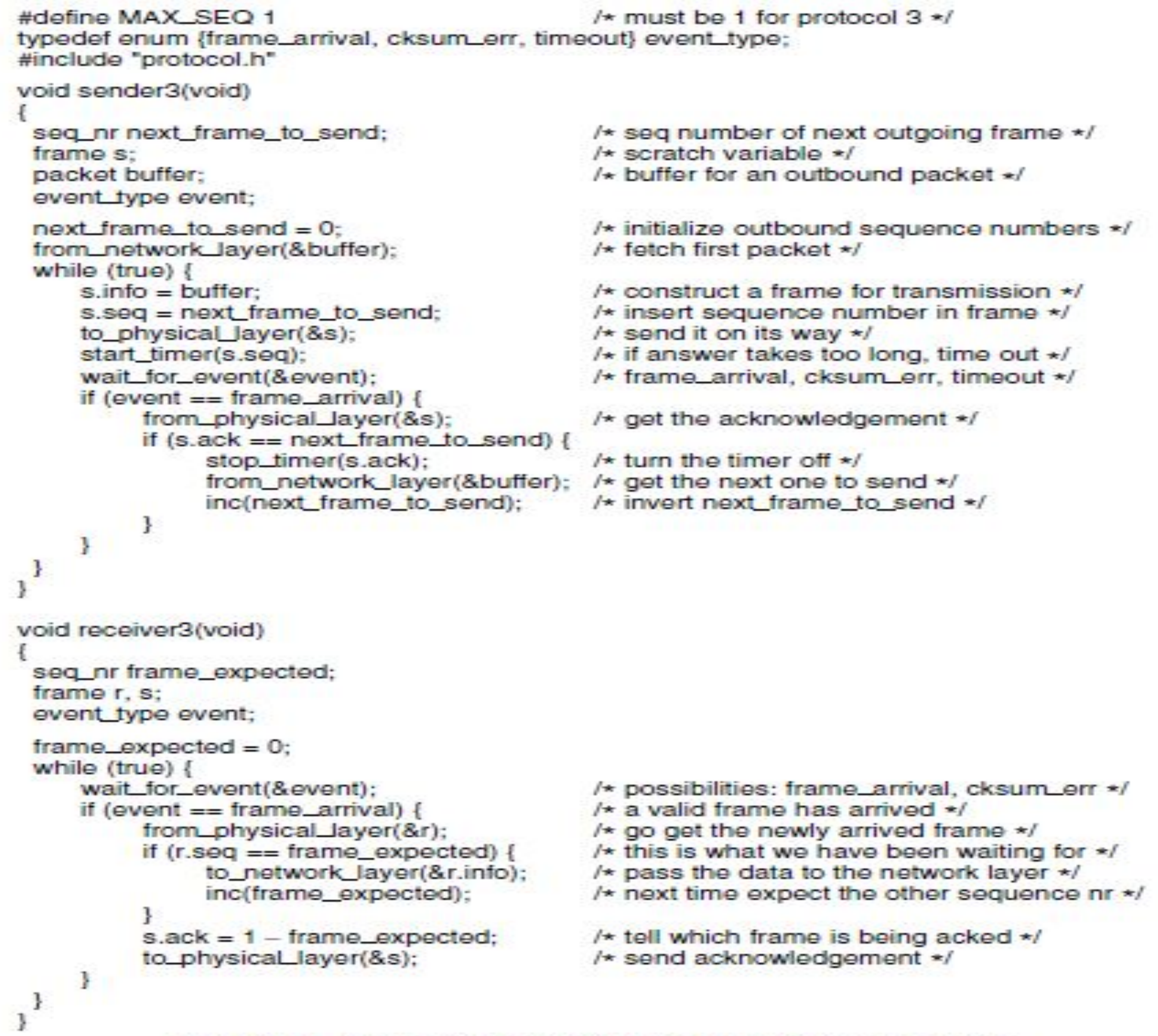
```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;                     /* frame_arrival is the only possibility */

  while (true) {
       from_network_layer(&buffer);     /* go get something to send */
       s.info = buffer;                 /* copy it into s for transmission */
       to_physical_layer(&s);           /* bye-bye little frame */
       wait_for_event(&event);          /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                           /* buffers for frames */
  event_type event;                     /* frame_arrival is the only possibility */
  while (true) {
       wait_for_event(&event);          /* only possibility is frame_arrival */
       from_physical_layer(&r);         /* go get the inbound frame */
       to_network_layer(&r.info);       /* pass the data to the network layer */
       to_physical_layer(&s);           /* send a dummy frame to awaken sender */
  }
}
```

Figure 3-13. A simplex stop-and-wait protocol.

A Simplex Stop-and-Wait Protocol for a Noisy Channel

- Stop – and – wait Automatic Repeat Request (Stop – and – Wait ARQ) is a variation of the above protocol with added error control mechanisms, appropriate for noisy channels.
- The sender keeps a copy of the sent frame. It then waits for a finite time to receive a positive acknowledgement from receiver.
- If the timer expires or a negative acknowledgement is received, the frame is retransmitted.
- If a positive acknowledgement is received then the next frame is sent.
- Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called **ARQ (Automatic Repeat reQuest) or PAR (Positive Acknowledgement with Retransmission**

- **Sender:**
- After transmitting a frame, the sender starts the timer running.
- If a valid ack comes in, the sender fetches the next packet from its network layer and puts it in the buffer, overwriting the previous packet.
- Advances the sequence number.
- If timer expires, neither the buffer nor the sequence number is changed
- **Receiver:**
- When a valid frame arrives at the receiver, its sequence number is checked to see if it is a duplicate.
- If not, it is accepted, passed to the network layer, and an acknowledgement is generated.



```
#define MAX_SEQ 1                                    /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
 seq_nr next_frame_to_send;                          /* seq number of next outgoing frame */
 frame s;                                            /* scratch variable */
 packet buffer;                                      /* buffer for an outbound packet */
 event_type event;

 next_frame_to_send = 0;                             /* initialize outbound sequence numbers */
 from_network_layer(&buffer);                        /* fetch first packet */
 while (true) {
      s.info = buffer;                               /* construct a frame for transmission */
      s.seq = next_frame_to_send;                    /* insert sequence number in frame */
      to_physical_layer(&s);                         /* send it on its way */
      start_timer(s.seq);                            /* if answer takes too long, time out */
      wait_for_event(&event);                        /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
           from_physical_layer(&s);                  /* get the acknowledgement */
           if (s.ack == next_frame_to_send) {
                stop_timer(s.ack);                   /* turn the timer off */
                from_network_layer(&buffer);         /* get the next one to send */
                inc(next_frame_to_send);             /* invert next_frame_to_send */
           }
      }
 }
}

void receiver3(void)
{
 seq_nr frame_expected;
 frame r, s;
 event_type event;

 frame_expected = 0;
 while (true) {
      wait_for_event(&event);                        /* possibilities: frame_arrival, cksum_err */
      if (event == frame_arrival) {                  /* a valid frame has arrived */
           from_physical_layer(&r);                  /* go get the newly arrived frame */
           if (r.seq == frame_expected) {            /* this is what we have been waiting for */
                to_network_layer(&r.info);           /* pass the data to the network layer */
                inc(frame_expected);                 /* next time expect the other sequence nr */
           }
           s.ack = 1 − frame_expected;               /* tell which frame is being acked */
           to_physical_layer(&s);                    /* send acknowledgement */
      }
 }
}
```

Figure 3-14. A positive acknowledgement with retransmission protocol.

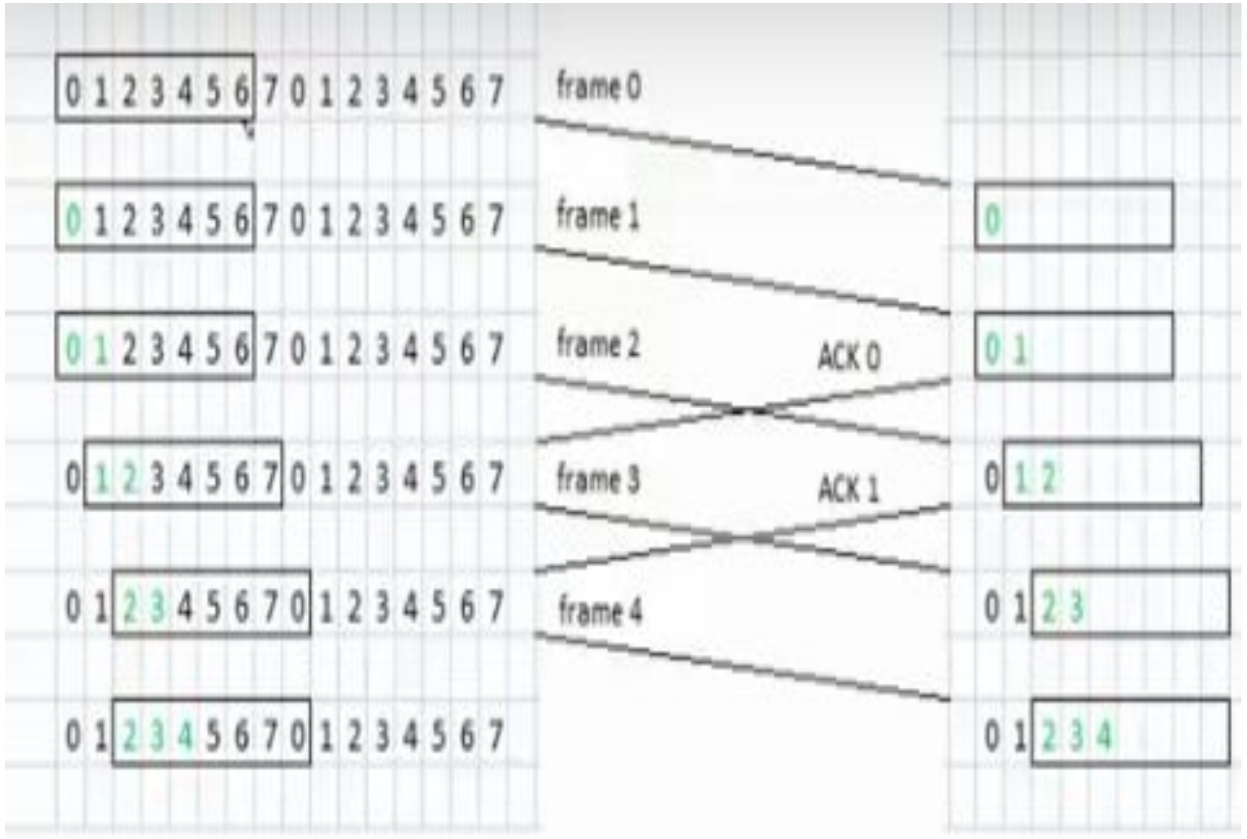
PiggyBacking

- When a data frame arrives, instead of immediately sending a separate control frame, the receiver waits until the network layer passes it the next packet.
- The acknowledgement is attached to the outgoing data frame (using the *ack field in the frame header). In effect, the acknowledgement gets a* free ride on the next outgoing data frame.
- The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking.**
- The principal advantage of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth.

Sliding Window Protocols

- The essence of all sliding window protocols is the sender maintains a set of sequence numbers corresponding to frames it is permitted to send.
- These frames are said to fall within the sending window. Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept.
- In some protocols the window size is fixed in size, but in others they can grow or shrink over the course of time as frames are sent and received.
- If the sequence number of the frames is an n-bit field, then the range of sequence numbers that can be assigned is 0 to 2^n-1. Consequently, the size of the sending window is 2^n-1.

Sliding Window for Sender & Reciever



- *The next three protocols are bidirectional protocols that belong to a class called sliding window protocols.*
- *The three differ among themselves in terms of efficiency, complexity, and buffer requirements.*
- *In these, as in all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum.*
- *The maximum is usually* 2^{n-1} *so the sequence number fits exactly in an n-bit field.*
- *The stop-and-wait sliding window protocol uses* $n = 1$*, restricting the sequence numbers to 0 and 1, but more sophisticated versions can use an arbitrary n.*

Sliding Window Protocols

- *A One-Bit Sliding Window Protocol*
- *A Protocol Using Go-Back-N*
- *A Protocol Using Selective Repeat*

One Bit Sliding Window Protocol

- The window size in this protocol is 1.
- Such a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.

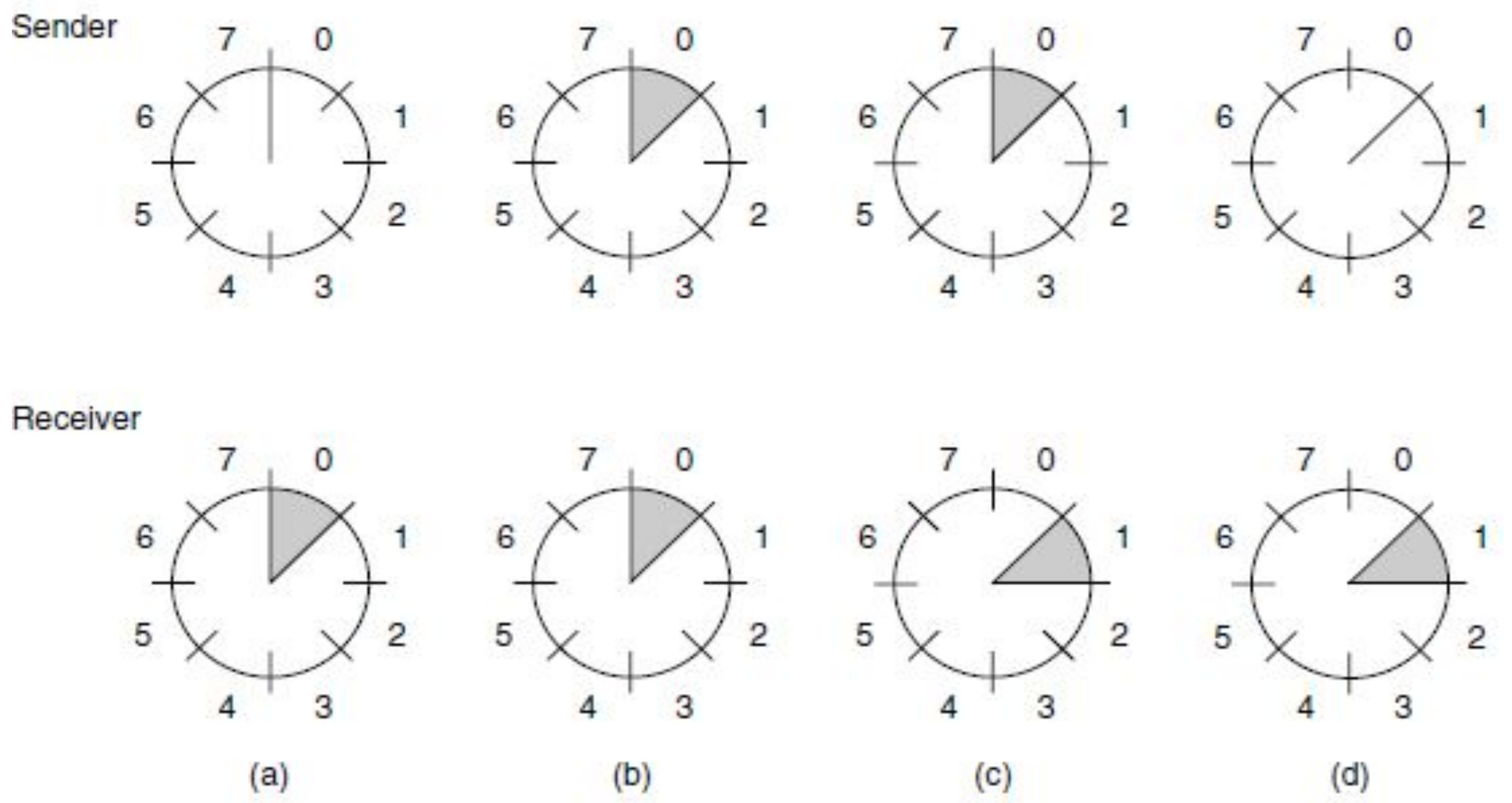
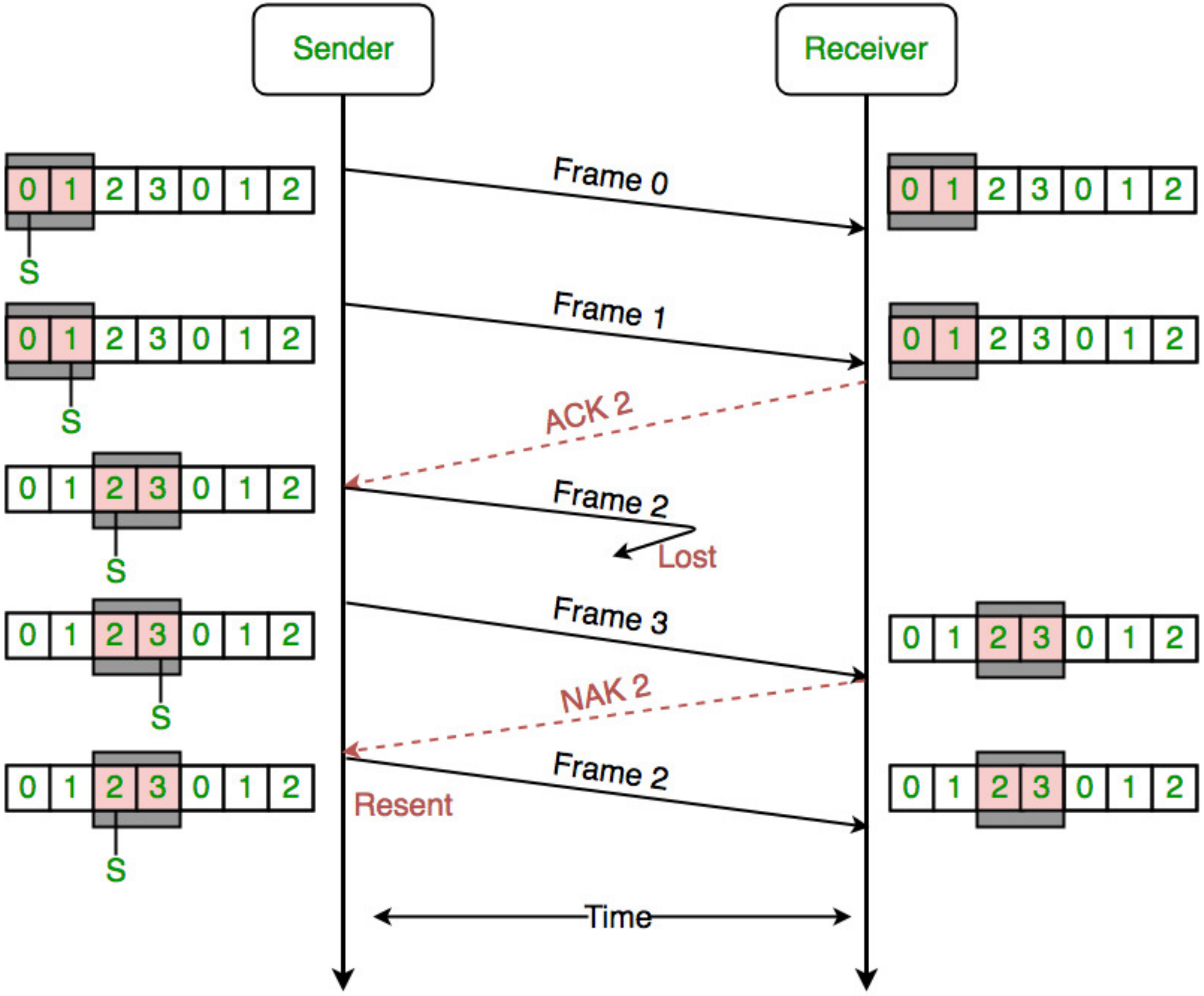


Figure 3-15. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

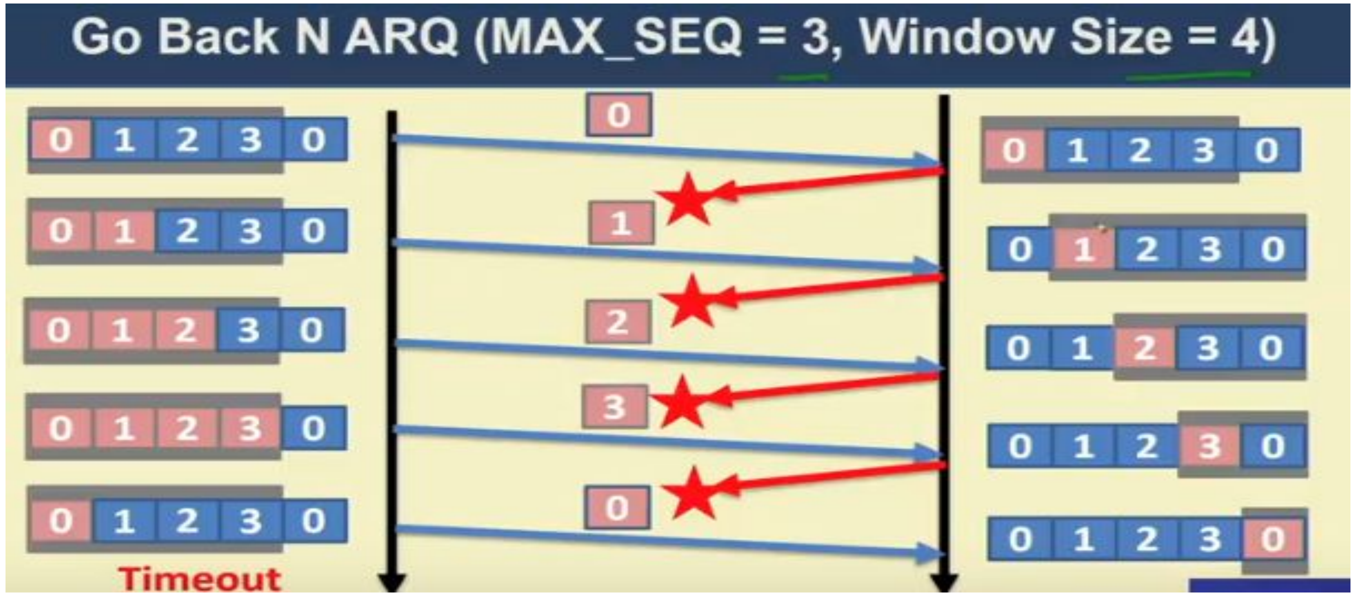
Go Back N

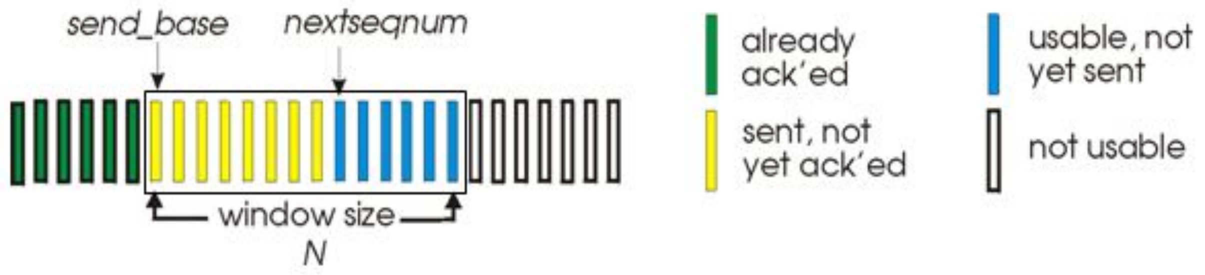
- It uses the concept of pipelining.
- Pipelining: A task is often begin before the previous task ended is known as pipelining(i.e. sending multiple frames before receiving the acknowledgment for the first frame.)
- The maximum number of frames that can be sent depends upon the size of the sending window. If the acknowledgment of a frame is not received within an agreed upon time period, all frames starting from that frame are retransmitted.

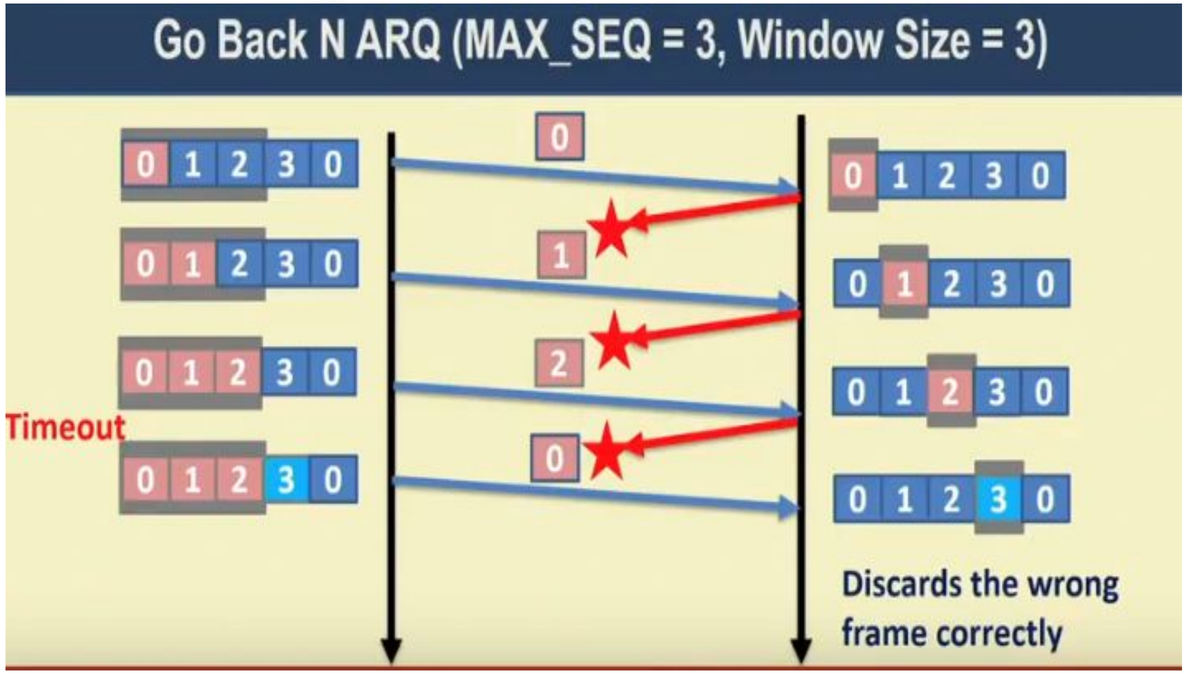

Sender
Receiver
Frame 0
Frame 1
ACK 2
Frame 2
Lost
Frame 3
NAK 2
Frame 2
Resent
Time
S
0 1 2 3 0 1 2

- Maximum number of outstanding frames(Window size)=MAX_SEQ
  Example: Sequence numbers(0,1,2..7)- 3 bit sequence numbers, number of outstanding frames(window size)=7(Not 8)

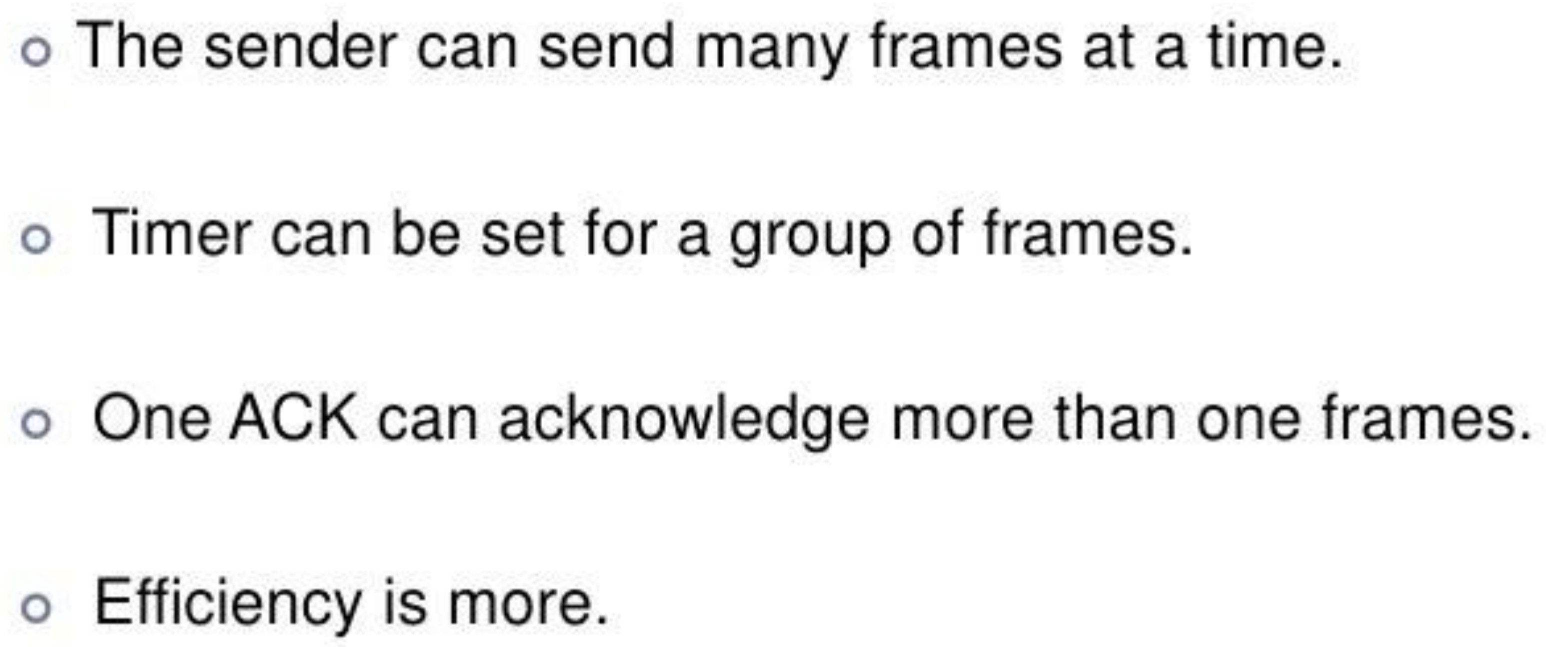
A Typical Example:




send_base
nextseqnum
window size
N
already ack'ed
sent, not yet ack'ed
usable, not yet sent
not usable


Go Back N ARQ (MAX_SEQ = 3, Window Size = 3)
0 1 2 3 0
0
1
2
0
Timeout
Discards the wrong frame correctly

Advantages



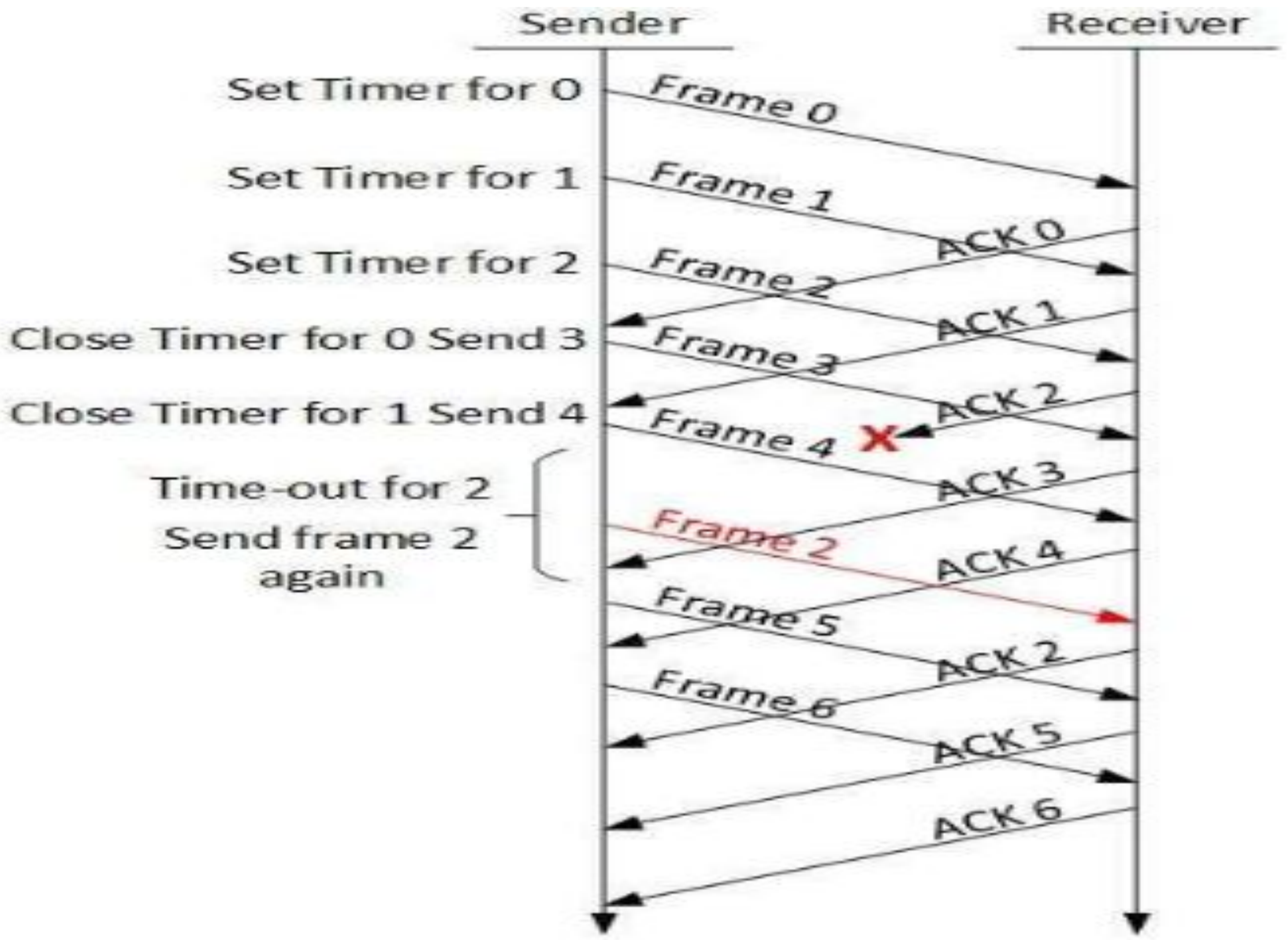
- The sender can send many frames at a time.
- Timer can be set for a group of frames.
- One ACK can acknowledge more than one frames.
- Efficiency is more.

DISADVANTAGES

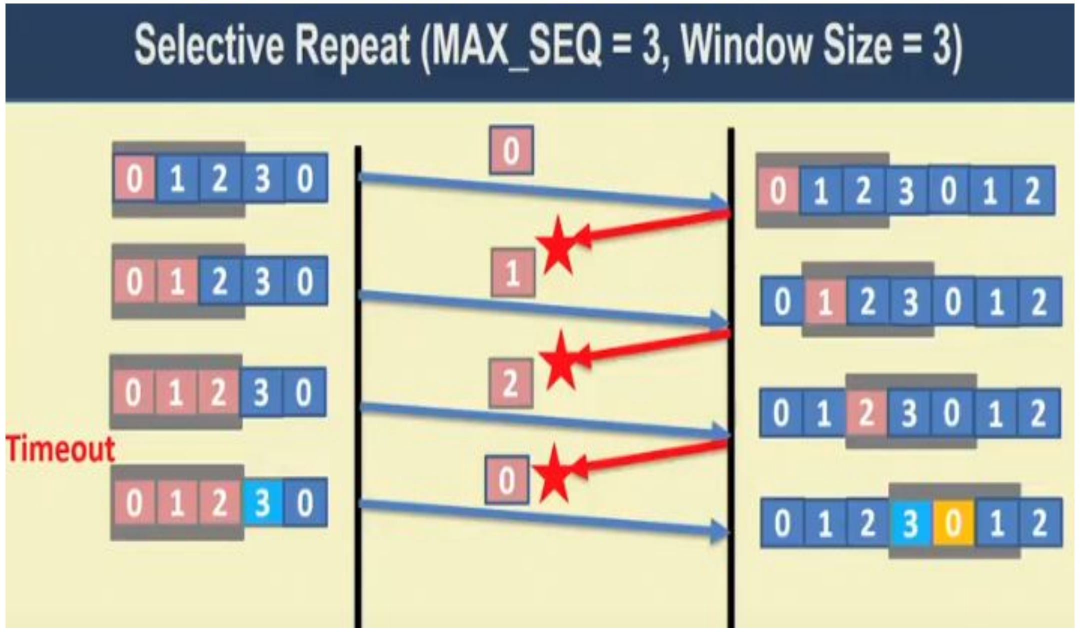
- Buffer requirement.
- Transmitter needs to store the last N packets.
- Scheme is inefficient when delay is large and data transmission rate is high.
- Unnecessary Retransmission of many error-free packets.

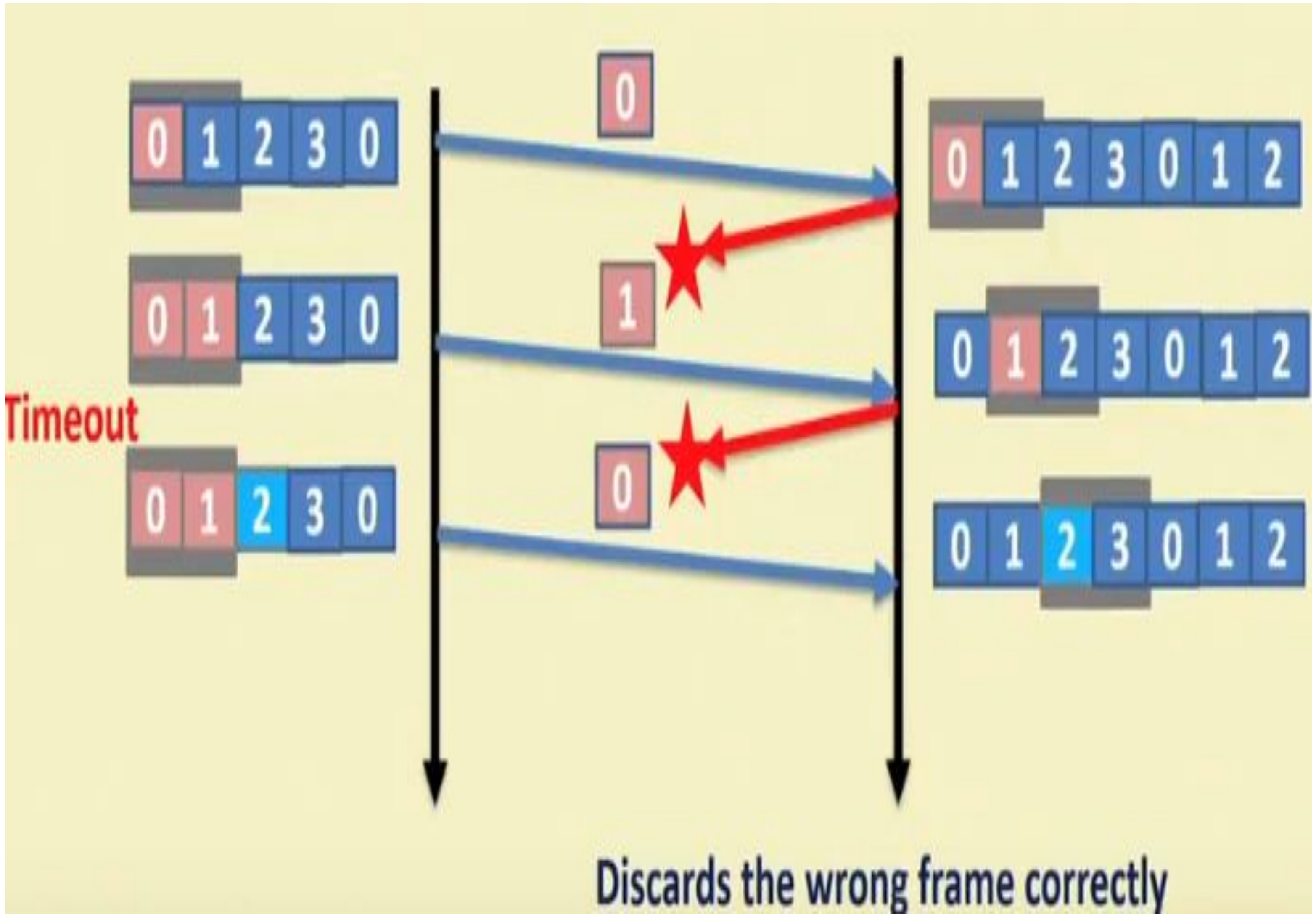
Selective Repeat

- The go-back-n protocol works well if errors are rare, but if the line is poor it wastes a lot of bandwidth on retransmitted frames.
- Selective Repeat protocol provides for sending multiple frames depending upon the availability of frames in the sending window, even if it does not receive acknowledgement for any frame.
- The receiver records the sequence number of the earliest incorrect or un-received frame. It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with ever acknowledgement frame.
- Once, Sender has sent all the frames in the window, it retransmits the frame whose sequence number is given by the acknowledgements. It then continues sending the other frames.

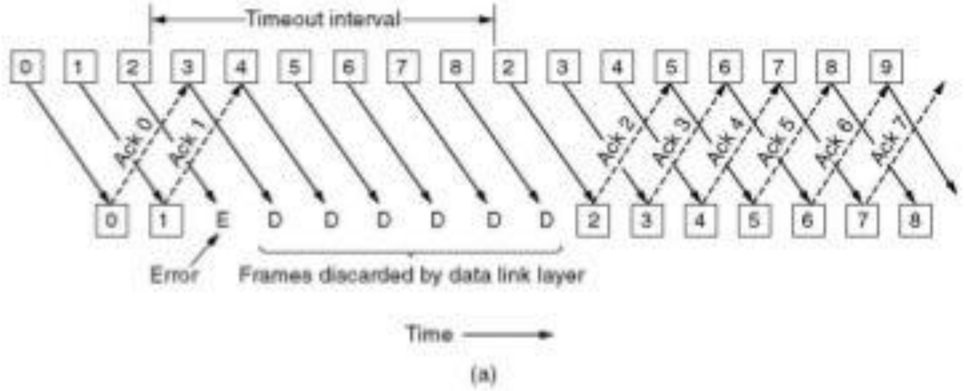

Sender
Receiver
Set Timer for 0
Set Timer for 1
Set Timer for 2
Close Timer for 0 Send 3
Close Timer for 1 Send 4
Time-out for 2
Send frame 2 again
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 2
Frame 5
Frame 6
ACK 0
ACK 1
ACK 2
ACK 3
ACK 4
ACK 2
ACK 5
ACK 6
X

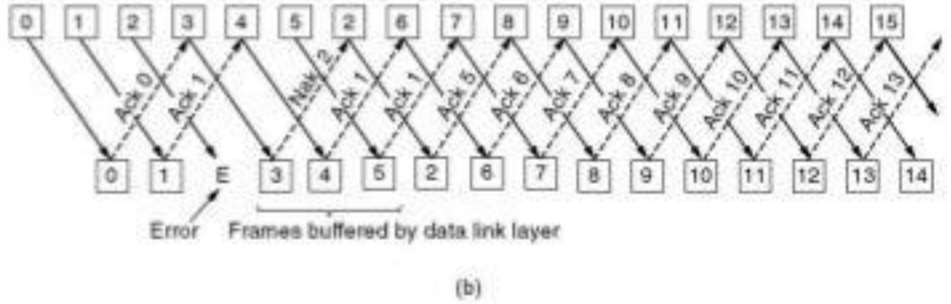
- Maximum number of outstanding frames(Window size)=(max_seq+1)/2.
- Example: Sequence numbers(0,1,2..7)- 3 bit sequence numbers, number of outstanding frames(window size)=4




0 1 2 3 0
0
0 1 2 3 0 1 2
1
0 1 2 3 0
0 1 2 3 0 1 2
Timeout
0
0 1 2 3 0
0 1 2 3 0 1 2
Discards the wrong frame correctly

Go-back-N Vs. Selective-repeat





Pipelining and error recovery. Effect on an error when

(a) Receiver's window size is 1.

(b) Receiver's window size is large.

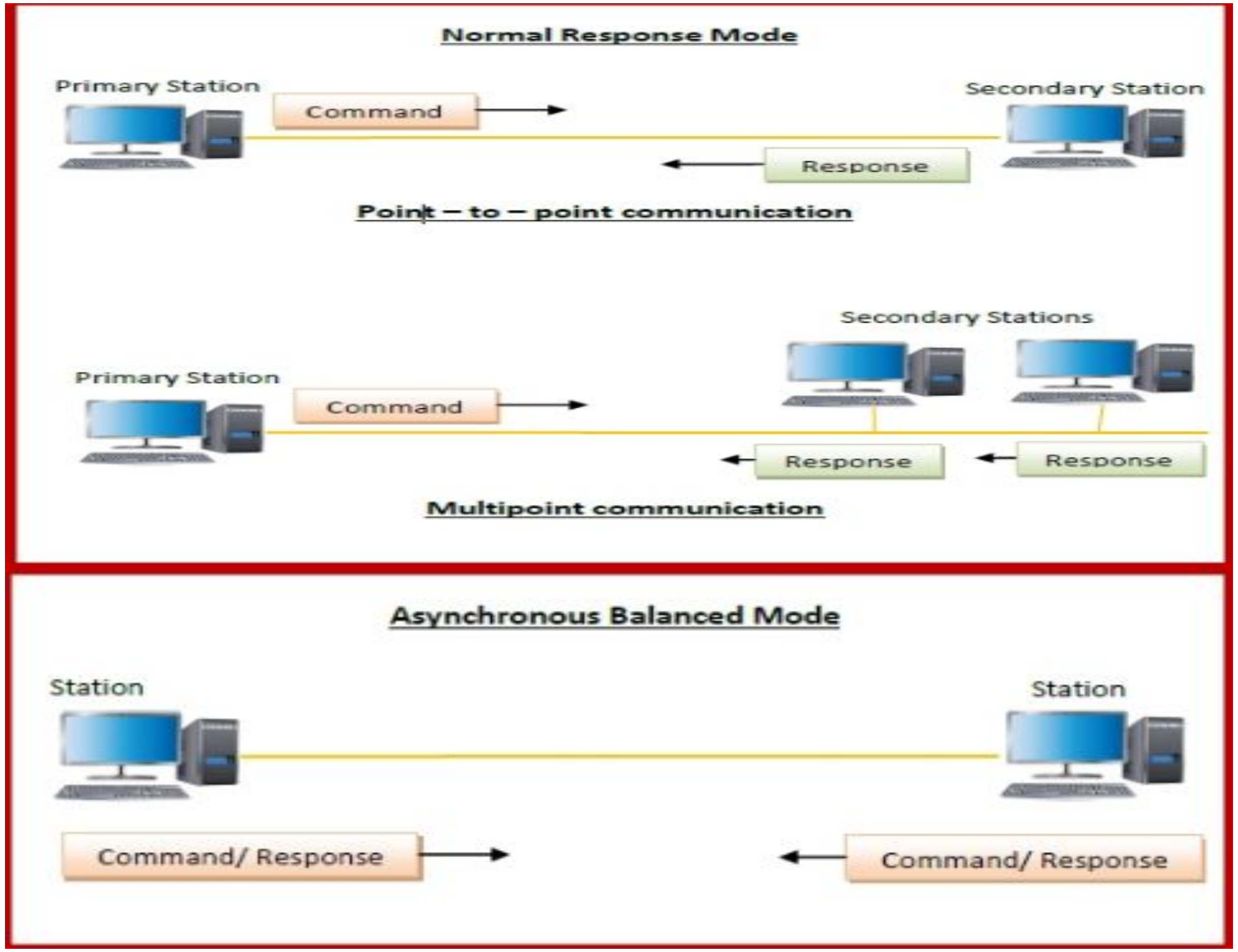
Data Link Layer in HDLC

HDLC

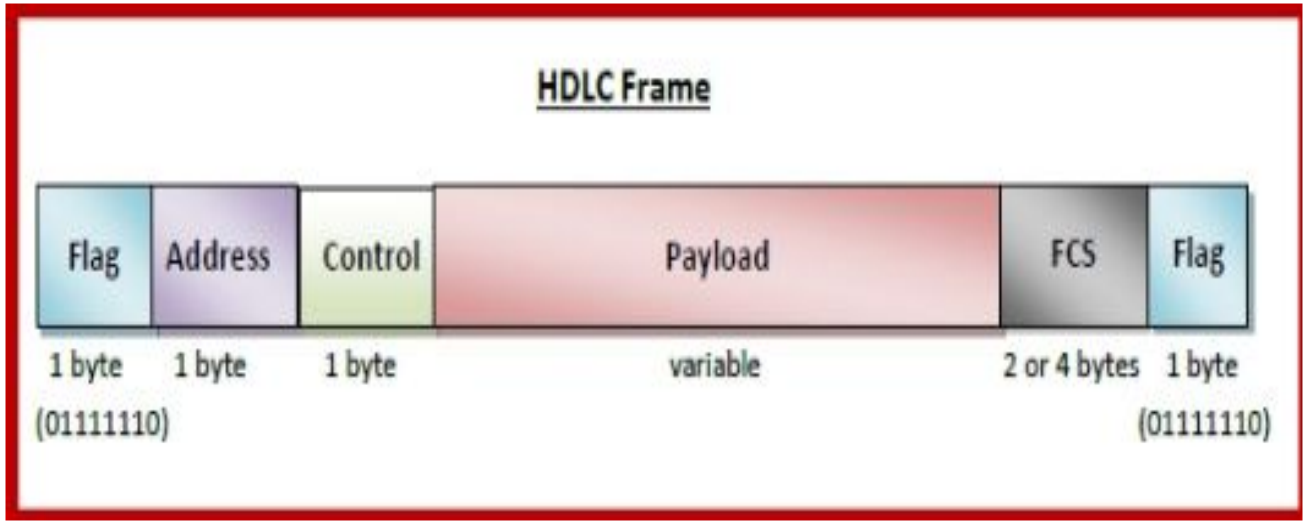
- High-level Data Link Control (HDLC) is a group of communication protocols of the data link layer for transmitting data between network points or nodes.
- It is a bit - oriented protocol that is applicable for both point - to - point and multipoint communications.

Transfer Modes:

- HDLC supports two types of transfer modes, normal response mode and asynchronous balanced mode.
- **Normal Response Mode (NRM)** − Here, two types of stations are there, a primary station that send commands and secondary station that can respond to received commands. It is used for both point - to - point and multipoint communications


Normal Response Mode
Primary Station
Command
Secondary Station
Response
Point – to – point communication
Secondary Stations
Primary Station
Command
Response
Response
Multipoint communication
Asynchronous Balanced Mode
Station
Station
Command/ Response
Command/ Response

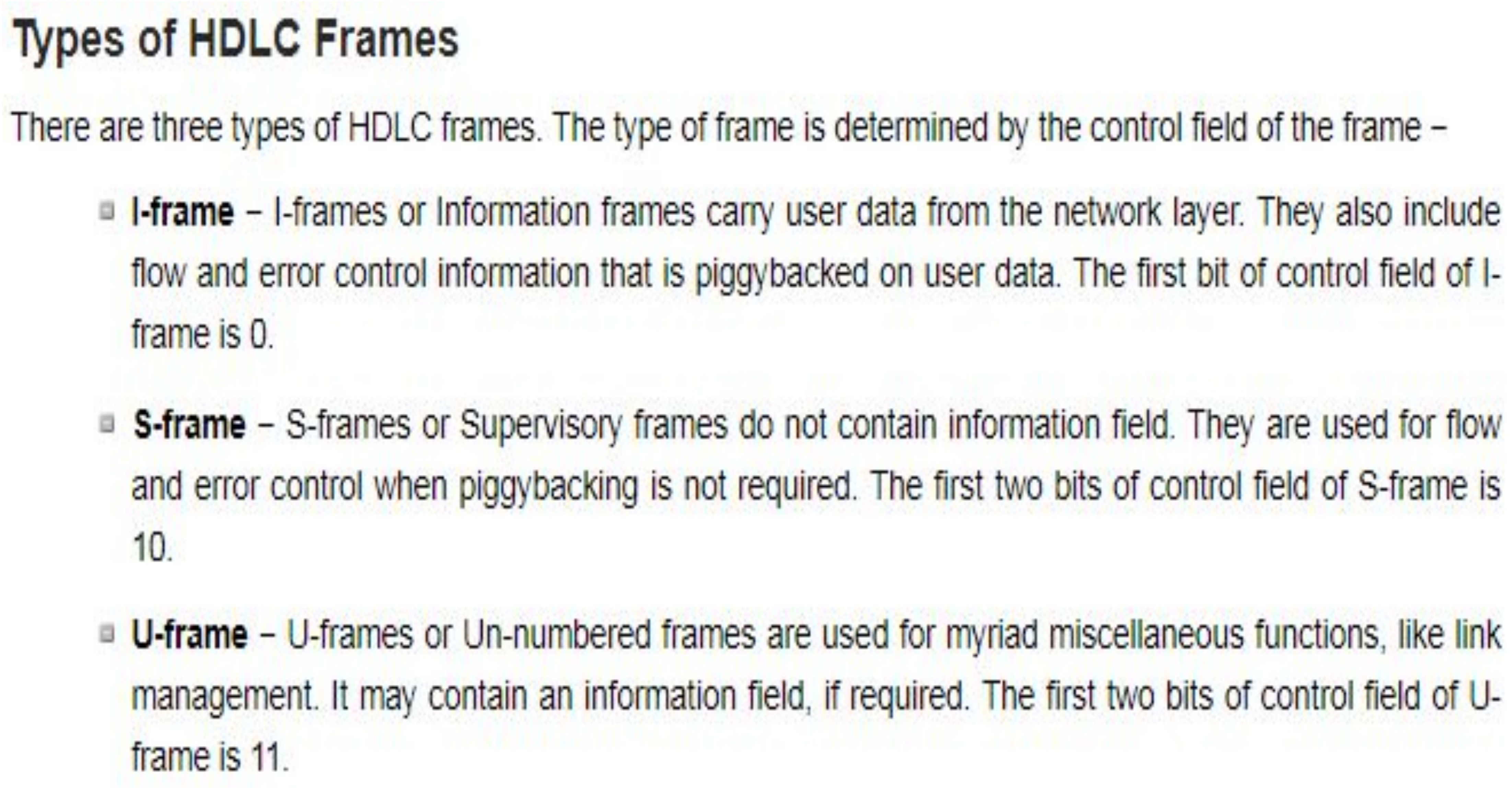
- **Asynchronous Balanced Mode (ABM)** − Here, the configuration is balanced, i.e. each station can both send commands and respond to commands. It is used for only point - to - point communications.
- **HDLC Frame:**



HDLC Frame

HDLC is a bit - oriented protocol where each frame contains up to six fields. The structure varies according to the type of frame. The fields of a HDLC frame are –

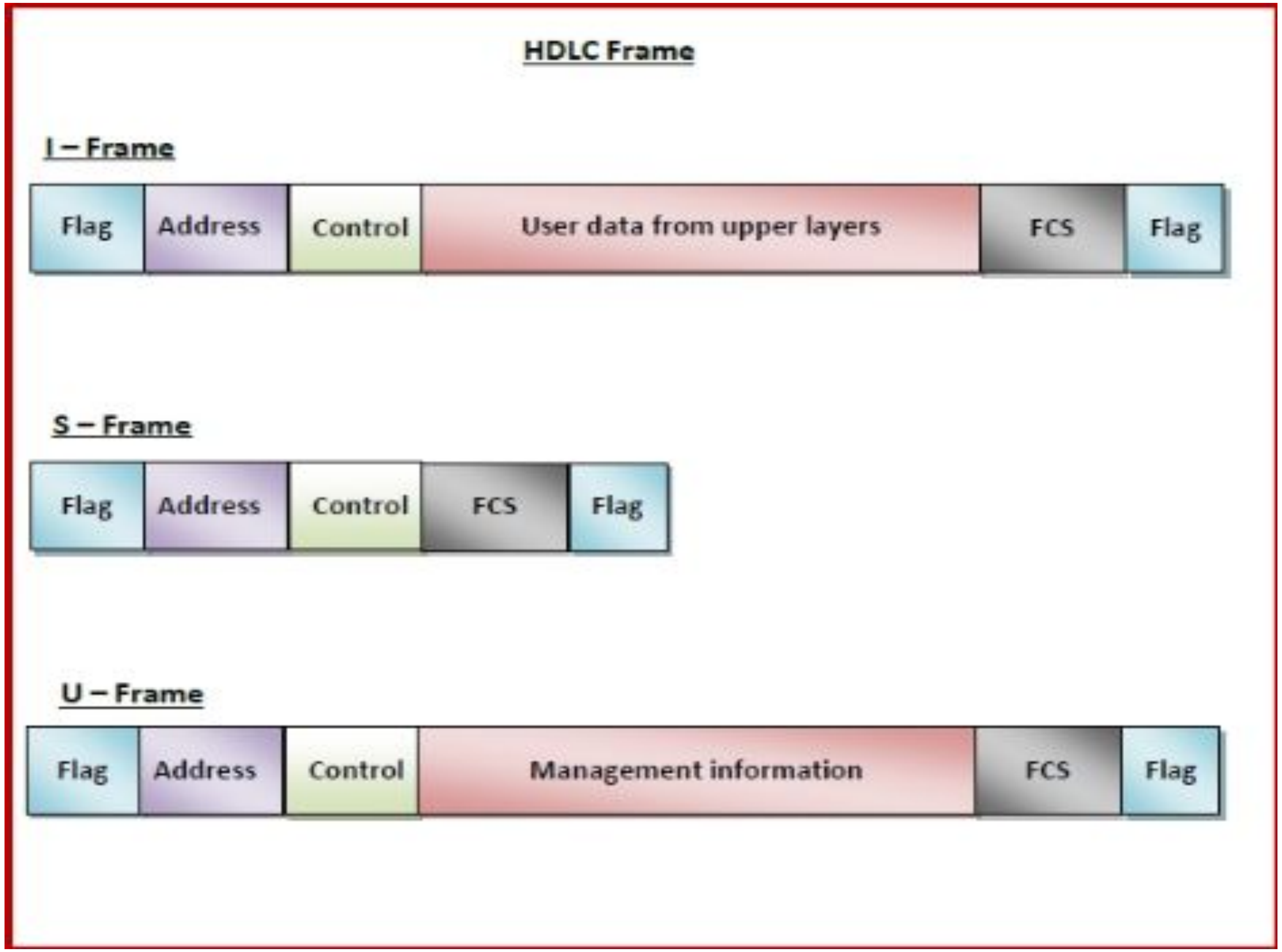
- **Flag** – It is an 8-bit sequence that marks the beginning and the end of the frame. The bit pattern of the flag is 01111110.
- **Address** – It contains the address of the receiver. If the frame is sent by the primary station, it contains the address(es) of the secondary station(s). If it is sent by the secondary station, it contains the address of the primary station. The address field may be from 1 byte to several bytes.
- **Control** – It is 1 or 2 bytes containing flow and error control information.
- **Payload** – This carries the data from the network layer. Its length may vary from one network to another.
- **FCS** – It is a 2 byte or 4 bytes frame check sequence for error detection. The standard code used is CRC (cyclic redundancy code)



Types of HDLC Frames

There are three types of HDLC frames. The type of frame is determined by the control field of the frame –

- **I-frame** – I-frames or Information frames carry user data from the network layer. They also include flow and error control information that is piggybacked on user data. The first bit of control field of I-frame is 0.
- **S-frame** – S-frames or Supervisory frames do not contain information field. They are used for flow and error control when piggybacking is not required. The first two bits of control field of S-frame is 10.
- **U-frame** – U-frames or Un-numbered frames are used for myriad miscellaneous functions, like link management. It may contain an information field, if required. The first two bits of control field of U-frame is 11.


HDLC Frame
I – Frame
Flag
Address
Control
User data from upper layers
FCS
Flag
S – Frame
Flag
Address
Control
FCS
Flag
U – Frame
Flag
Address
Control
Management information
FCS
Flag

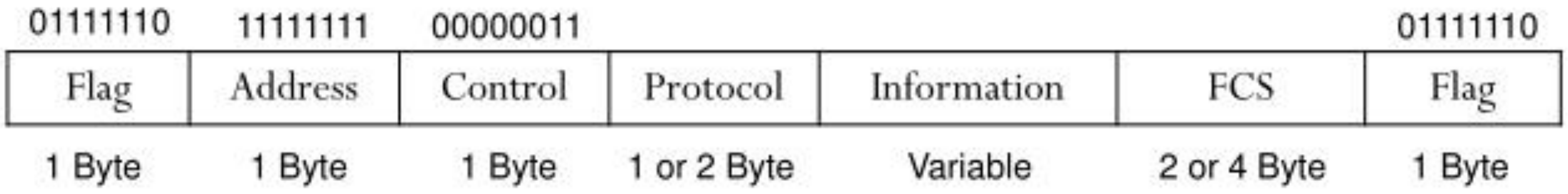
PPP (Point to Point Protocol)

- *The PPP frame format was chosen to closely resemble the frame format of HDLC (High-level Data Link Control), a widely used instance of an earlier family of protocols, since there was no need to reinvent the wheel.*
- *The primary difference between PPP and HDLC is that PPP is byte oriented rather than bit oriented.*
- *In particular, PPP uses byte stuffing and all frames are an integral number of bytes.*
- *HDLC uses bit stuffing and allows frames of, say, 30.25 bytes.*

- *There is a second major difference in practice, however. HDLC provides reliable transmission with a sliding window, acknowledgements, and timeouts in the manner we have studied.*
- *PPP can also provide reliable transmission in noisy environments, such as wireless networks; the exact details are defined in RFC 1663.*

PPP Frame Format

- Frame format of PPP is similar to HDLC frame:

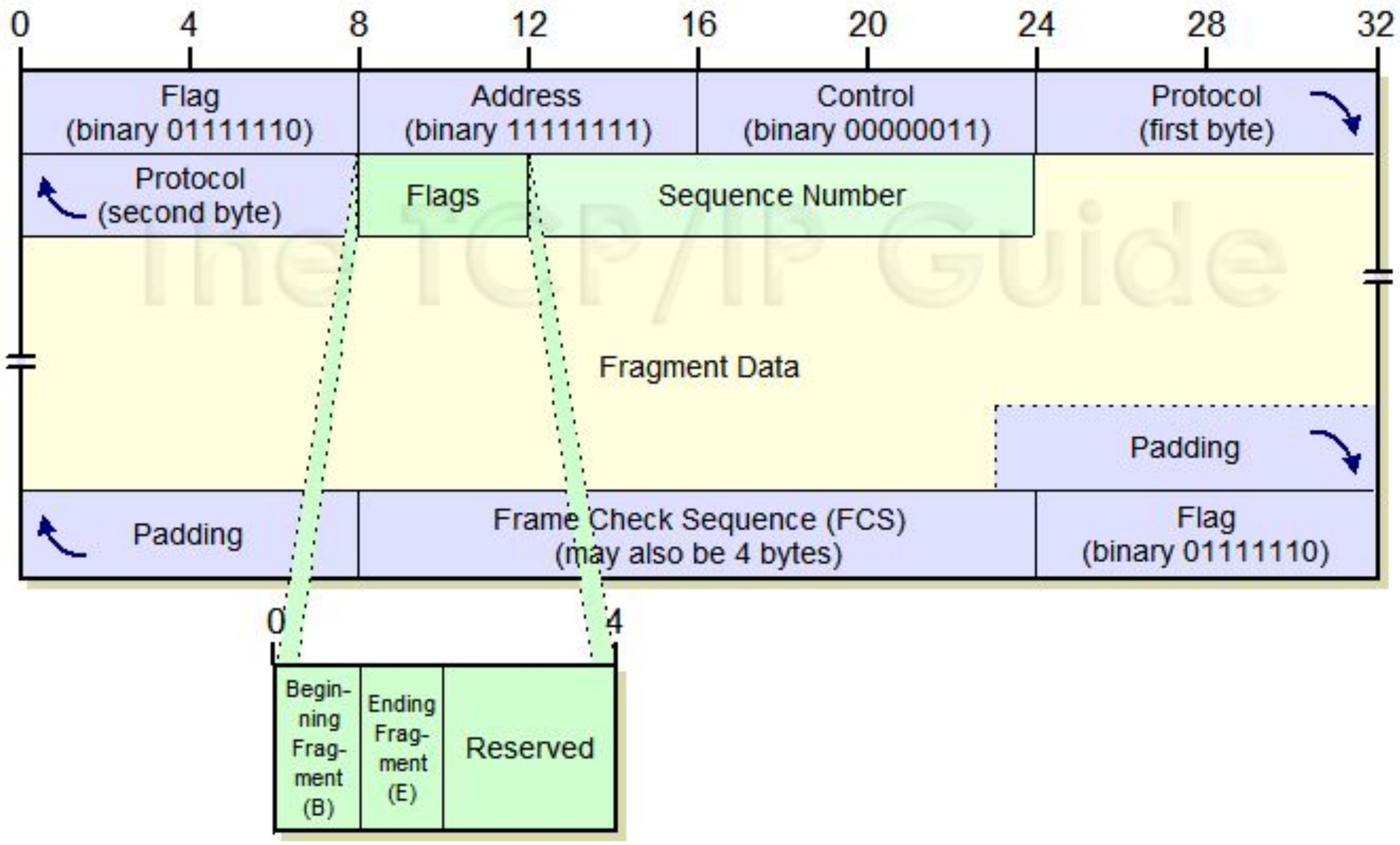


01111110	11111111	00000011				01111110
Flag	Address	Control	Protocol	Information	FCS	Flag
1 Byte	1 Byte	1 Byte	1 or 2 Byte	Variable	2 or 4 Byte	1 Byte

- **Control Field:** It is also of 1 byte. It uses the format of U-Frame in HDLC. The value is always 00000011 to show that the frame does not contain any sequence number and there is no flow control or error control.

- **Protocol Field:** This field specifies the kind of protocol of the data in the information field.

Multilink PPP



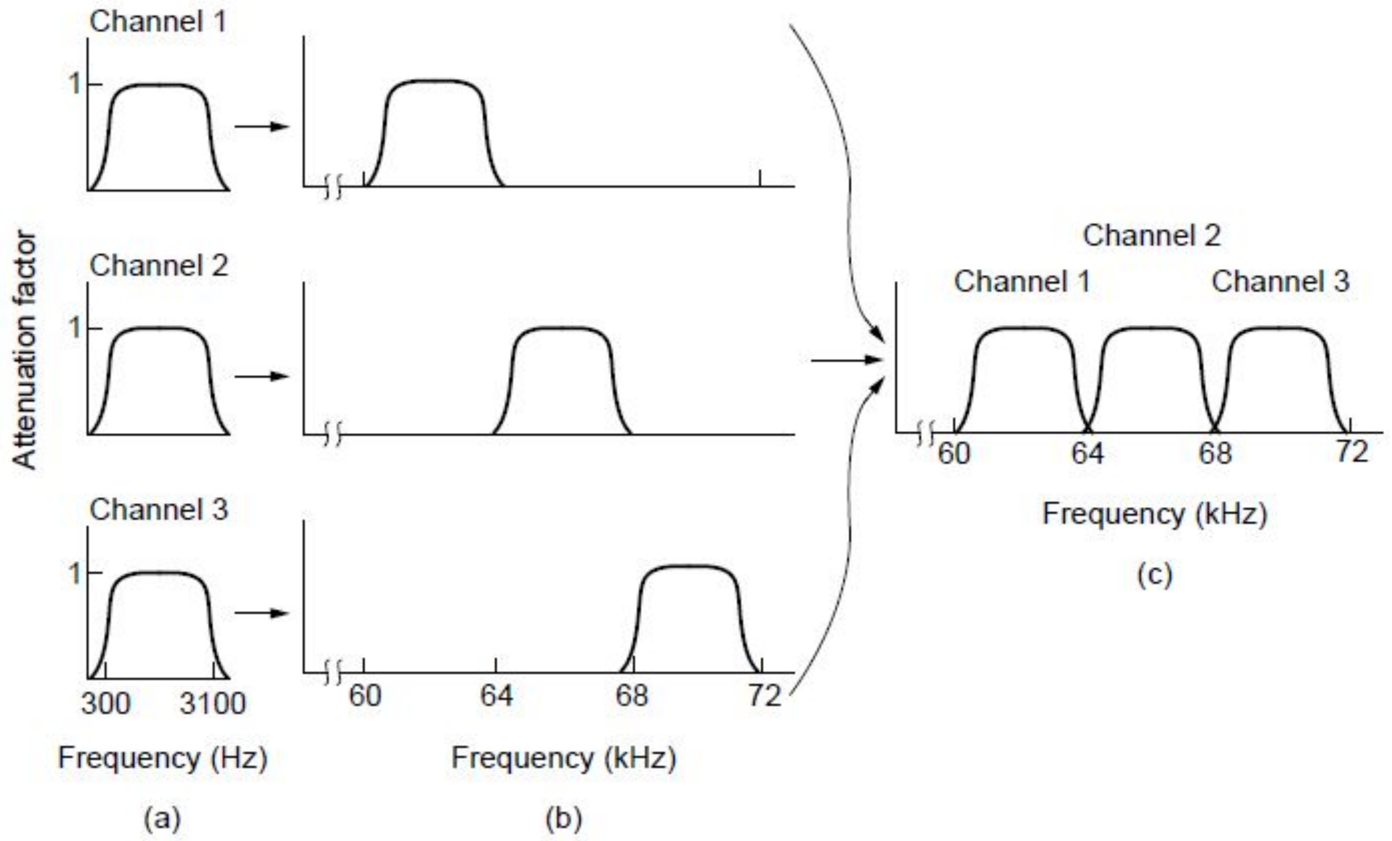
Field Name	Size (bytes)	Description
B	1/8 (1 bit)	***Beginning Fragment Flag:*** When set to 1, flags this fragment as the first of the split-up PPP frame. It is set to 0 for other fragments.
E	1/8 (1 bit)	***Ending Fragment Flag:*** When set to 1, flags this fragment as the last of the split-up PPP frame. It is set to 0 for other fragments.
Reserved	2/8 (2 bits) ***OR*** 6/8 (6 bits)	***Reserved:*** Not used, set to zero.
Sequence Number	1 1/2 (12 bits) ***OR*** 3 (24 bits)	***Sequence Number:*** When a frame is split up, the fragments are given consecutive sequence numbers so the receiving device can properly reassemble them.
Fragment Data	Variable	***Fragment Data:*** The actual fragment from the original PPP frame.

Multiplexing

- The process of sharing of channels by multiple signals
- Can use a single wire to carry several signals than to install a separate wire for every signal
- Frequency Division Multiplexing
- Time Division Multiplexing
- Code Division Multiplexing

FDM

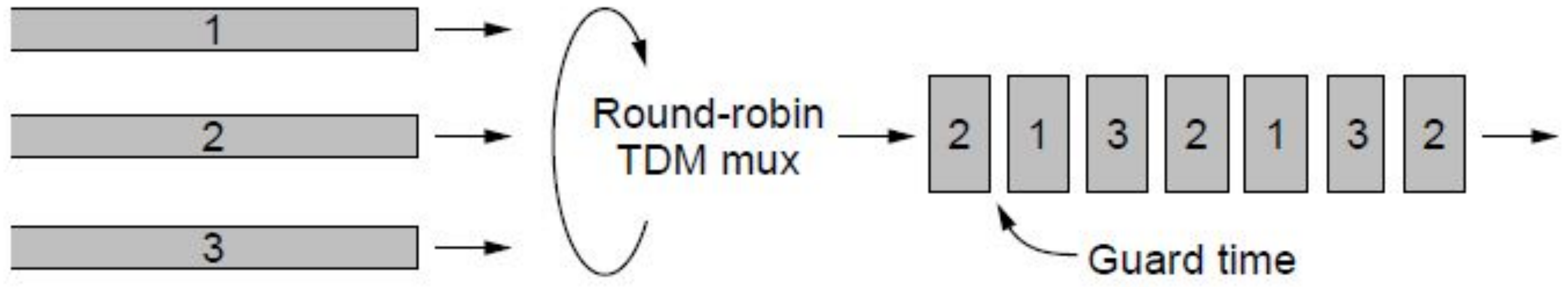
- takes advantage of passband transmission to share a channel
- divides the spectrum into frequency bands
- each user having exclusive possession of some band in which to send their signal
- AM : 500 – 1500kHz
- FM : 88 – 108 MHz



- Frequency division multiplexing. (a) The original bandwidths. (b) The bandwidths raised in frequency. (c) The multiplexed channel.

Time Division Multiplexing

- The users take turns (in a round-robin fashion), each one periodically getting the entire bandwidth for a little burst of time
- **Bits** from each input stream are **taken** in a **fixed time slot** and output to the aggregate stream
- the **streams** must be **synchronized** in time
- Small intervals of **guard time (**analogous to a frequency guard band) may be added to accommodate small timing variations


1
2
3
Round-robin TDM mux
2 1 3 2 1 3 2
Guard time

Code Division Multiplexing

- a form of **spread spectrum communication** in which a narrowband signal is spread out over a wider frequency band
- more tolerant of interference
- multiple signals from different users share the same frequency band

- Each station transmit over the entire frequency spectrum all the time.
- Multiple simultaneous transmissions are separated using **coding theory**.
- Analogy of airport lounge
- the key to CDMA is to be able to extract the desired signal while rejecting everything else as random noise

- each bit time is subdivided into *m short intervals called* ***chips***
- 64 or 128 chips per bit
- Each station is assigned a unique *m-bit code called* a **chip sequence**

- To transmit a 1 bit, a station sends its chip sequence
- To transmit a 0 bit, it sends the negation of its chip sequence

- Each station has its own unique chip sequence
- All chip sequences are pairwise **orthogonal**
- The normalized inner product of any two distinct chip sequences, **S and T (written as S T), is 0**
- The normalized inner product of any chip sequence with itself is 1
- **Walsh codes – chip sequences**

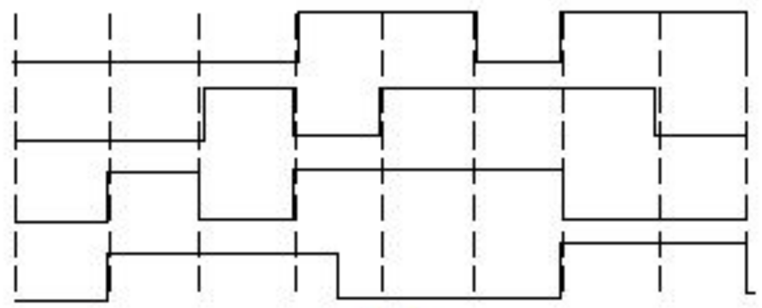
A = (−1 −1 −1 +1 +1 −1 +1 +1)

B = (−1 −1 +1 −1 +1 +1 +1 −1)

C = (−1 +1 −1 +1 +1 +1 −1 −1)

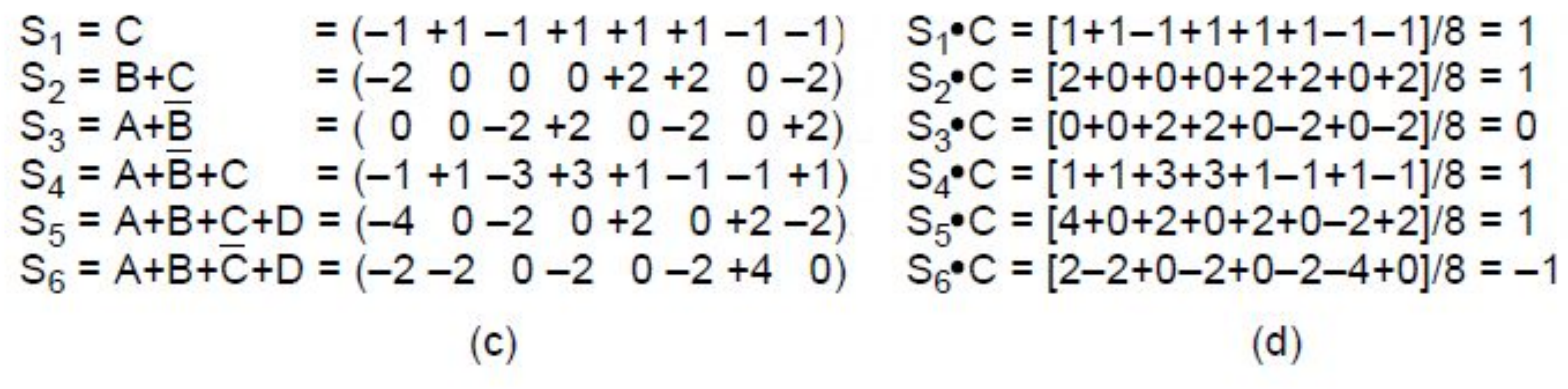
D = (−1 +1 −1 −1 −1 −1 +1 −1)

(a)


(b)

(a) Chip sequences for four stations.

(b) Signals the sequences represent



$$\begin{array}{ll}
S_1 = C & = (-1\ +1\ -1\ +1\ +1\ +1\ -1\ -1) \\
S_2 = B+C & = (-2\ \ 0\ \ 0\ \ 0\ +2\ +2\ \ 0\ -2) \\
S_3 = A+\overline{B} & = (\ 0\ \ 0\ -2\ +2\ \ 0\ -2\ \ 0\ +2) \\
S_4 = A+\overline{B}+C & = (-1\ +1\ -3\ +3\ +1\ -1\ -1\ +1) \\
S_5 = A+B+C+D & = (-4\ \ 0\ -2\ \ 0\ +2\ \ 0\ +2\ -2) \\
S_6 = A+B+\overline{C}+D & = (-2\ -2\ \ 0\ -2\ \ 0\ -2\ +4\ \ 0)
\end{array}$$

(c)

$$\begin{array}{l}
S_1 \bullet C = [1+1-1+1+1+1-1-1]/8 = 1 \\
S_2 \bullet C = [2+0+0+0+2+2+0+2]/8 = 1 \\
S_3 \bullet C = [0+0+2+2+0-2+0-2]/8 = 0 \\
S_4 \bullet C = [1+1+3+3+1-1+1-1]/8 = 1 \\
S_5 \bullet C = [4+0+2+0+2+0-2+2]/8 = 1 \\
S_6 \bullet C = [2-2+0-2+0-2-4+0]/8 = -1
\end{array}$$

(d)

(a) Six examples of transmissions.

(b) Recovery of station C's