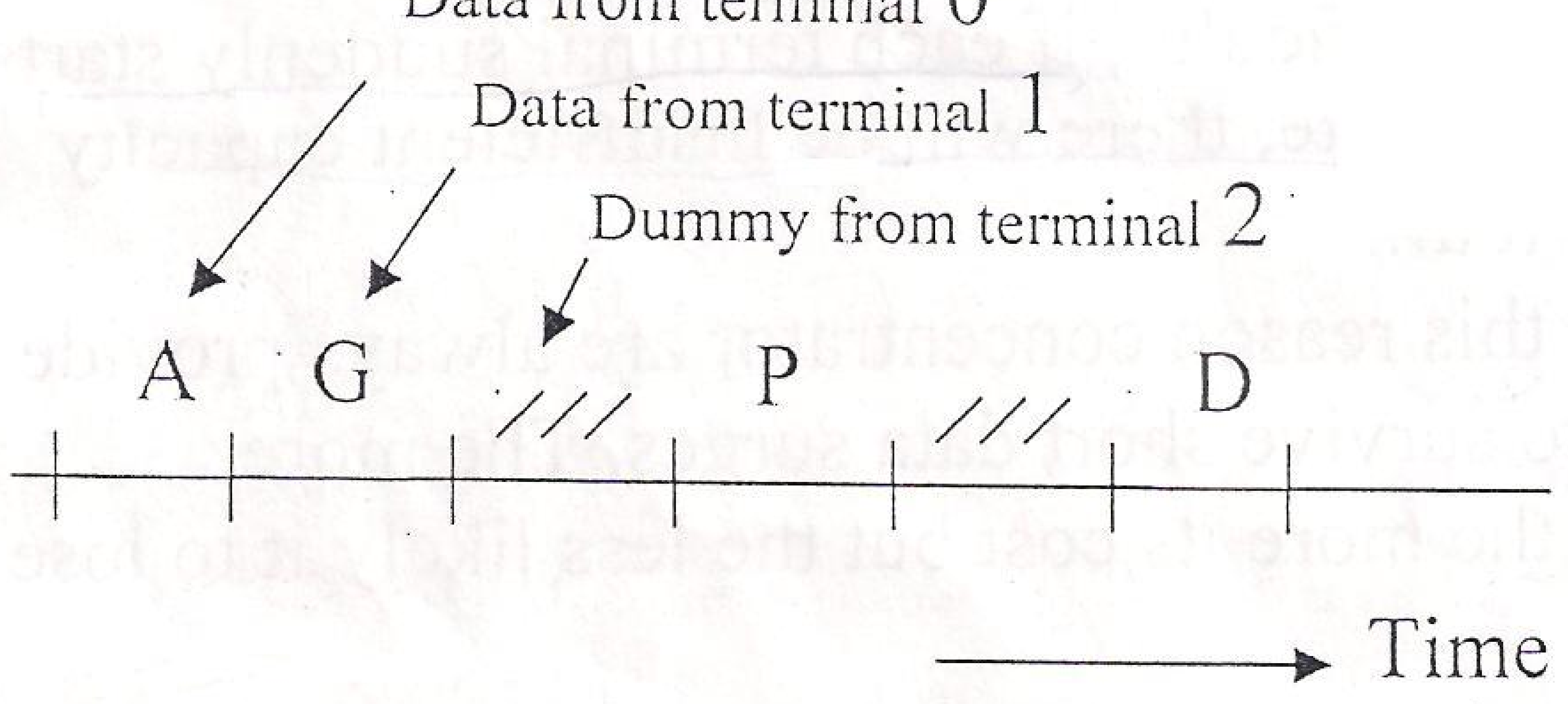
The advantage here is that it is not necessary to keep turning the line around just to discover that a terminal has nothing to say.
For the case of a (star controller) Roll Call polling is often used any way to allow the master to acquire input in an orderly fashion. These poll Messages differs from those of the multidrop case because there are no site address needed, a terminal only receives those polls directed to it.

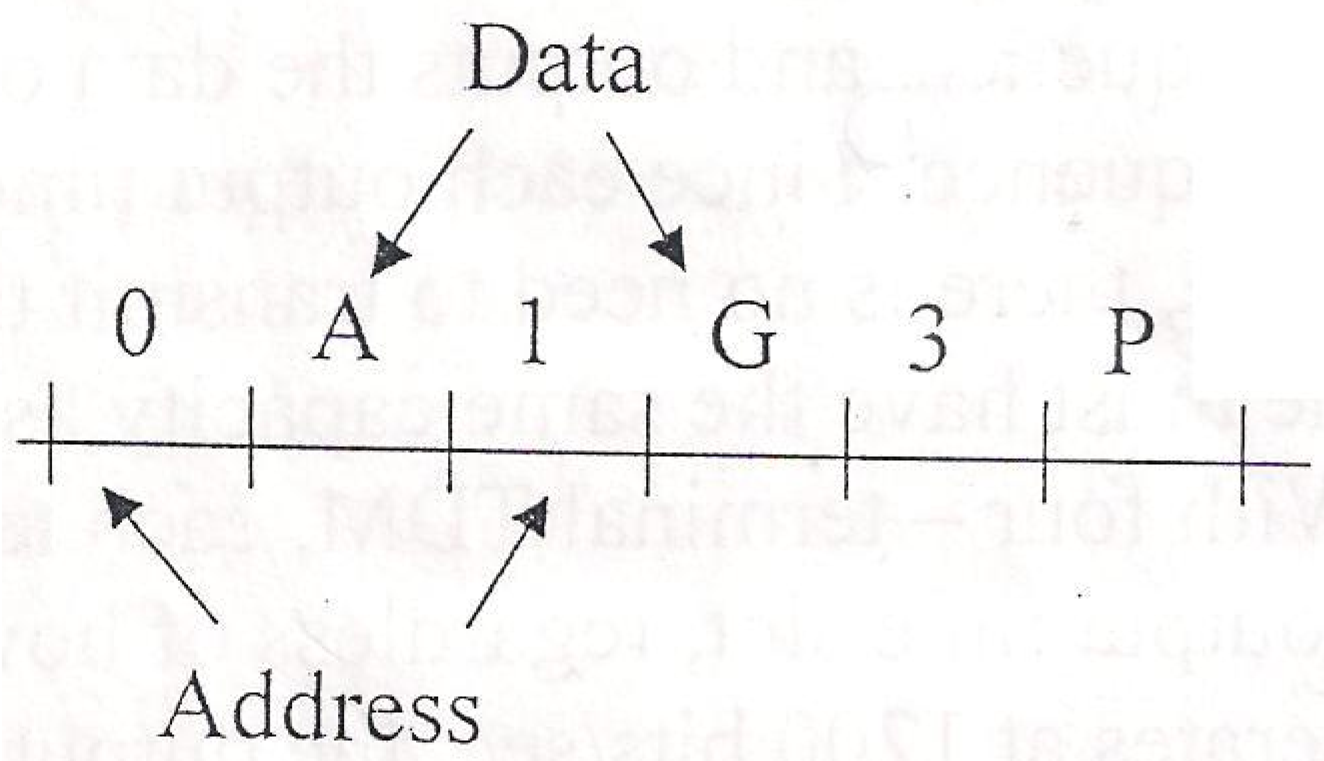
Multiplexing Versus Concentration

Terminal controllers can be divided into two general classes, multiplexers and concentrators.
A multiplexer: - is a device that accept, input from a collection of lines in some static, predetermined sequence, and outputs the data onto a single output line in the same sequence. Since each output time slot is dedicated to a specific input line, there is no need to transmit the input lines numbers .The output line must have the same capacity as the sum of the input line capacities. With four – terminal TDM, each terminal is allocated one – fourth of the output time slot, regardless of how busy it is, if each of the terminals operates at 1200 bits/sec, the output line must be (4 * 1200 = 4800 bits/sec), since four characters must be send during **each polling cycle .**



The big disadvantage of TDM is that when a terminal has no traffic, an output time slot is wasted. Initially the multiplexer and the computer

Synchronize themselves. Both know that the order to be used, for example, 0123 0123. The data, themselves carry no identification of their origin. If each terminal has traffic only a small fraction of the time, TDM makes inefficient use of the output line capacity. When the actual traffic is far below the potential traffic, most of the time slots on the output line is wasted.
Consequently it is often possible to use an output line with less capacity than the sum of the input lines. This arrangement is called concentration The usual approach is to only transmit actual data and not dummy characters. However, this strategy introduces the problem of telling the receiver which character come from which input line.
One Solution to this problem is to send two output characters for each input Character; the terminal number and the data.



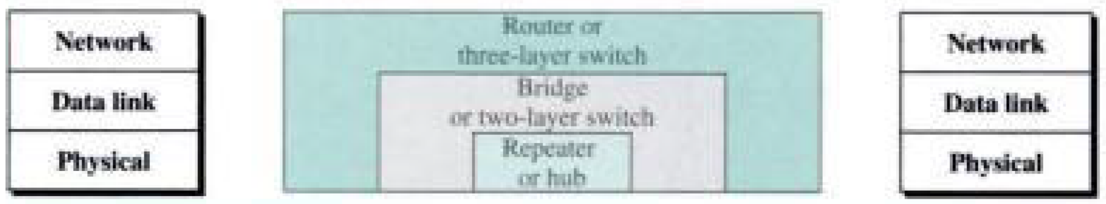
Four terminal concentrations

Concentrators using this principle are often referred to as statistical Multiplexer or ATDM (A synchronous time division multiplexer)
Concentration has an inherent difficulty, if each terminal suddenly starts outputting data at its maximum rate, there will be insufficient capacity on the output line to handle it all.
Some data may be lost. For this reason concentrator are always provide with extra buffers in order to survive short data surges. The more memory a concentrator has the more its cost but the less likely it to lose data.

CONNECTING DEVICES

In this section, we divide connecting devices into five different categories based on the layer in which they operate in a network, as shown in Figure below:

e



T h fi v

e categories contain devices which can be defined as
1. Those which operate **below** the physical layer such as a passive hub.
2. Those which operate **at** the physical layer (a repeater or an active hub).
3. Those which operate **at** the physical and data link layers (a bridge or a two-layer switch).
4. Those which operate **at** the physical, data link, and network layers (a router or a three-layer switch).
5. Those which can operate **at** all five layers (a gateway).

Passive Hubs

- A passive hub is just a connector. It connects the wires coming from different branches.
- This type of a hub is part of the media.
- Its location in the Internet model is below the physical layer.

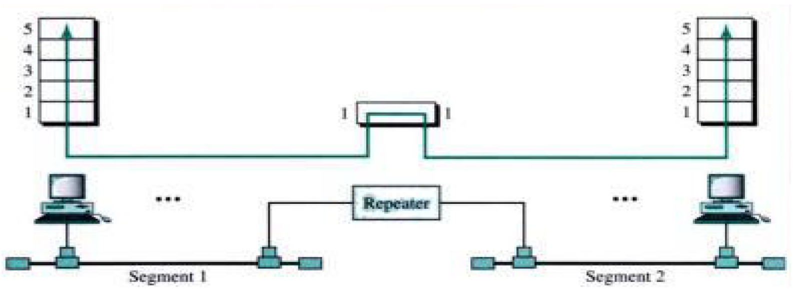
Repeaters

- A **repeater** is a device that operates only in the physical layer.
- Signals that carry information within a network can travel a fixed distance before attenuation endangers the integrity of the data. A repeater receives a signal and, before it becomes too weak or corrupted, **regenerates** the original bit pattern. The repeater then sends the **refreshed signal**.
- A repeater does not actually connect two LANs; it connects two segments of the **same LAN**. The segments connected are still part of **one single LAN.**

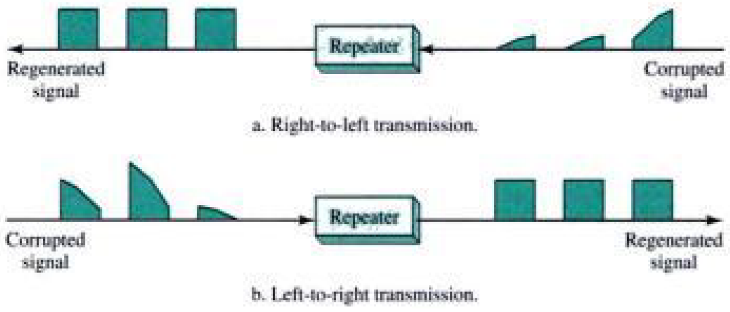
- A repeater is not a device that can connect two LANs of different protocols.
- It is important to compare a repeater to an amplifier, but the comparison is inaccurate:

1. An amplifier cannot discriminate between the intended signal and noise; it amplifies equally everything fed into it.
2. A repeater does not amplify the signal; it regenerates the signal. When it receives a weakened or corrupted signal, it creates a copy, bit for bit, at the original strength.

- A repeater can extend the physical length of a LAN, as shown in Figure below:

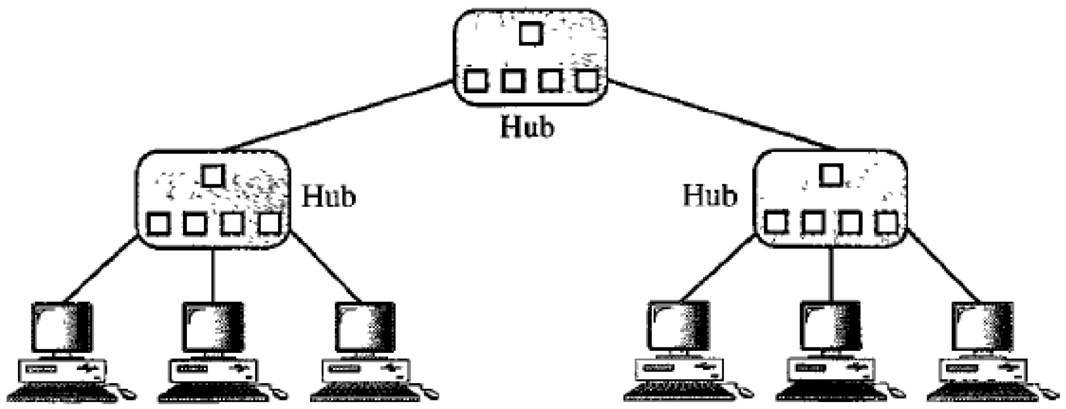


- **The location of a repeater** on a link is vital. A repeater must be placed so that a signal reaches it before any noise changes the meaning of any of its bits. A little noise can alter the precision of a bit's voltage without destroying its identity (see Figure below). If the corrupted bit travels much farther, however**, accumulated noise** can change its meaning completely. At that point, the original voltage is **not recoverable**, and the error needs to be corrected. A repeater placed on the line before the legibility of the signal becomes lost can still read the signal well enough to determine the intended voltages and replicate them in their original form.



Active Hubs

- An **active hub** is actually **a multipart repeater**.
- It is normally used to create connections between stations in a physical star topology.
- hubs can also be used to create multiple levels of hierarchy, as shown in Figure below:



- What is the difference in functionality between a **bridge** and a **repeater**?
- Bridge has **filtering capability**. It can check the **destination address** of a frame and decide if the frame should be **forwarded or dropped**. If the frame is to be forwarded, the decision must specify the port. A bridge has a **table** that maps addresses to ports.

Two-Layer Switches

- We can have a **two-layer switch or a three-layer switch**.
- A **three-layer switch** is used at the network layer; it is a kind of router.
- The **two-layer switch** performs at the physical and data link layers.
- A **two-layer switch** is a **bridge, a bridge** with many ports and a design that allows better (faster) performance.
- A bridge with a few ports can connect a few LANs together.
- A bridge with many ports may be able to allocate a unique port to each station, with each station on its own independent entity.

Routers

- A router is a **three-layer device** that routes packets based on their **logical addresses** (host-to-host addressing).
- A **router** normally connects LANs and WANs in the Internet and has a routing table that is used for making decisions about the route.

<u>Three-Layer Switches</u>

- **A three-layer switch is a router**, but a **faster and more sophisticated**.
- The switching fabric in a three-layer switch allows faster table lookup and forwarding.

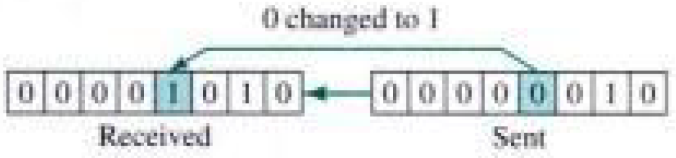
Gateway

- A gateway is normally a computer that operates in all five layers of the Internet or seven layers of OSI model.
- A gateway takes an application message, reads it, and interprets it. This means that it can be used as a connecting device between two internetworks that use different models.
- **For example**, a network designed to use the OSI model can be connected to another network using the Internet model.
- The gateway connecting the two systems can take a frame as it arrives from the first system, move it up to the OSI application layer, and remove the message. Gateways can provide security.

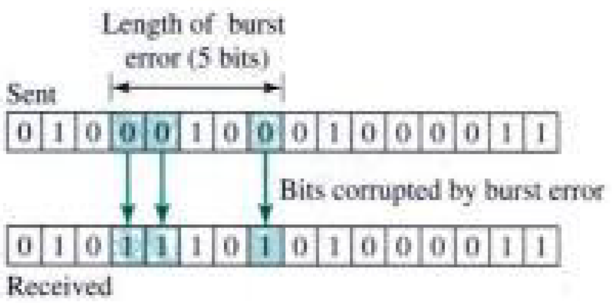
Error Detection and Correction

Types of Errors:

- **Single bit error:** it means that only one bit of a given data unit such as byte, character, data unit, or packet, is changed from 1 to 0 or from 0 to 1.



- Single bit errors are the least likely type of error in serial data transmission. To understand why, imagine a sender sends data at a rate of 1 Mbps, this means that each bit lasts only 1/1000000 sec. For a single bit error to occur the noise must have duration of only 1µsec, which is very rare, noise normally lasts much longer than this.
- Single bit error can occur in parallel data transmission. For example, if we are sending 8 bits of 1 byte at the same time, and one of the wires is noisy, one bit can be corrupted in each byte.
- Burst error: It means that to or more bits in the data unit have changed from 1 to 0 or from 0 to 1.
- The length of the burst is measured from the first corrupted bit to the last corrupted bit, some bits in between may have not been corrupted.
-



- Burst error is must likely to occur in a serial transmission. The duration of noise is normally longer than the duration of one bit, which means that when noise affects data, it affects a set of bits. The number of bits affected depends on **data rate and duration of noise.**

- For example, if we are sending data at 1 Kbps, a noise of 1/100 sec can affect 10 bits; if we are sending data at 1 Mbps, the same noise can affect 10000 bits.

Error detection

- Before correcting the error, we must detect it.
- Redundancy: error detection using the concept of redundancy means adding extra bits for detecting errors at the destination.
- Three types of redundancy checks are common in data communication: **parity check, cyclic redundancy check (CRC), and checksum.**

Parity Check:

- It consists of tow types, **Simple parity check and two dimensional parity check**.
- **Simple Parity Check:** in this technique, a redundant bit called a parity bit is added to every data unit so that the total number of ones in the unit (including the parity bit) becomes even (or odd).
- **Example:** suppose we want to send the following data units, find the suitable parity bit for each unit: 1110111 1101111 1110010 1101100 1100100
- **Solution:** (Whenever the total numbers of ones is even: add 1 to the data unit, and for odd number of ones, add 0 to the data unit).

1110111<u>**0**</u> 1101111<u>**0**</u> 1110010<u>**0**</u> 1101100<u>**0**</u> 1100100<u>**1**</u>

- Performance: simple parity check can detect all single bit errors.

- It also can detect burst errors as long as the total number of bits changed is odd (1, 3, 5, …etc). let's say we have an even parity data unit where the total number of ones, including the parity bit, is **6:1000111011**. If any 3 bits change value, the resulting parity will be odd and the error will be detected: **1*111*111011:9**, ***011*0111011:7**, **1*1*00*0*1*0*011:5** all odd. The checker would return a result of 1, and the data unit would be rejected. The same holds true for any odd number of errors.

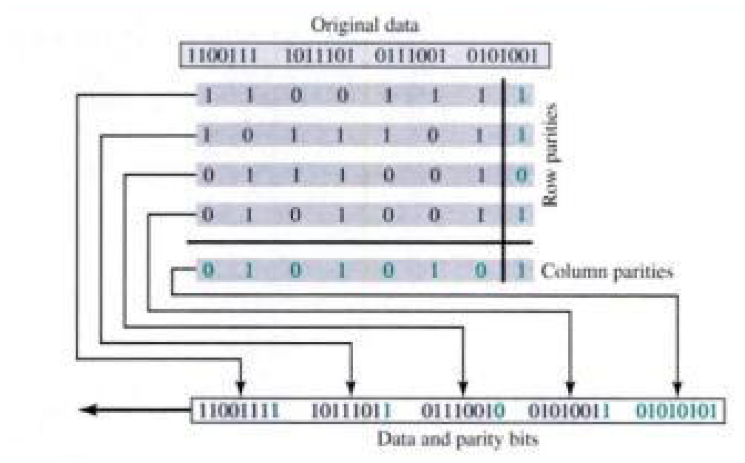
- Now, suppose that 2 bits of the data unit are changed: **1*11*0111011:8, 1*1*00*0*11011:6, 1000*0*1101*0*:4.** In each case the number of ones in the data unit is still even. The parity checker will add them and return an even number although the data unit contains two errors. **So, this method cannot detect errors where the total number of bits changed is even.**

- **Two dimensional parity check:** In this method, a block of bits is organized in a table (rows and columns). Then we calculate the parity bit for each data unit. After that we organize them into a table. Finally, we attach the generated parity bit unit with the original data to be transmitted.

- **Example:** if we have four data units to be transmitted, calculate the parity bit using two dimensional technique?

- **Solution:** be arranging it in a table, we will have four rows and eight columns. We then calculate the parity bit for each column and create a new row of eight bits; they are the parity bits for the whole block.

- **Note:** the first parity bit in the fifth row is calculated based on all first bits; the second is calculated based on all the second bits, and so on.



- Example: suppose the following block is sent:

  10101001 00111001 11011101 11100111 10101010

- However, it is hit by a burst noise of length 8, and some bits are corrupted.

  10100011 10001001 11011101 11100111 10101010

- When the receiver checks the parity bits, some of the bits do not follow the even parity rule and the whole block is discarded (note that the non-matching bits are in bold):

  10100011 10001001 11011101 11100111 **1**0**101**0**1**0

- Two dimensional parity check increases the probability of detecting burst errors.

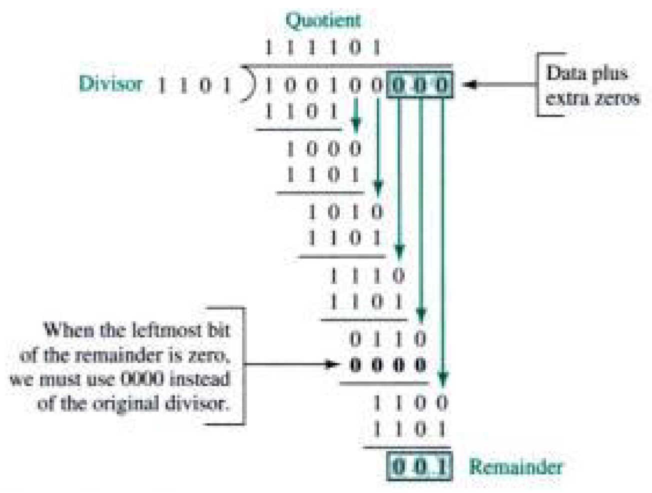
- There is however one pattern of errors that remain elusive. If 2 bits in one data unit are damaged and two bits in exactly the same positions in another data unit are also damaged, the checker will not detect an error.

- Consider, for example, two data units: 11110000 and 11000011. If the first and last bits in each of them are changed, making the data units read 01110001 and 01000010, the errors cannot be detected by this method.

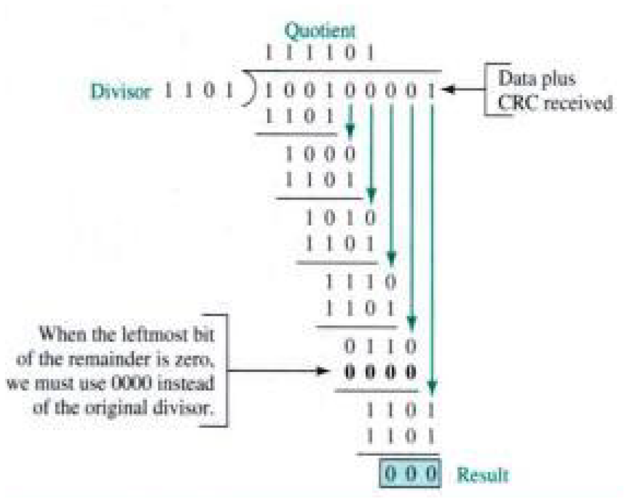
Cyclic Redundancy Check (CRC):

- Unlike the parity check which is based on addition, CRC is based on binary division. In CRC, instead of adding bits to achieve a desired parity, a sequence of redundant bits called the CRC **or the CRC reminder**, is appended to the end of a data unit so that the resulting data unit becomes divisible by a second, predetermined binary number.
- At its destination, the incoming data unit is divided by the same number. If at this step there is no reminder, the data unit is assumed to be intact and is therefore accepted.
- A reminder indicates that the data unit has been damaged in transit and therefore must be rejected.
- The redundancy bits used by CRC are derived by dividing the data unit by predetermined divisor; the reminder is the CRC.
- A CRC must have two qualities: it must have exactly one bit less than the divisor, and appending it to the data string must make the resulting bit sequence exactly divisible by the divisor.
- The following steps clarify the process of deriving the CRC:
- **First:** a string of **n** zeros is appended to the data unit. The number n is 1 less than the number of bits in the predetermined divisor, which is n+1 bits.
- **Second:** the newly elongated data unit is divided by the divisor, using a process called binary division. The reminder resulting from this division is the CRC.
- **Third:** the CRC of n bits derived in step2 replaces the appended zeros at the end of the data unit. Note that the CRC may consist of all zeros.
- The data unit arrives at the receiver, data first followed by the CRC. The receiver treats the whole string as a unit and divides it by the same divisor that was used to find the CRC reminder.
- If the string arrives without errors, the CRC checker yields a reminder of zero and the data unit passes

- If the string has been changed in transit, the division yields a non-zero reminder and the data unit doesn't pass.
- **The CRC generator:** it uses module-2 division, first, a 4-bit divisor is subtracted from the first 4-bits of the dividend. Each bit in the divisor is subtracted from the corresponding bit of the dividend without disturbing the next higher bit.
- In the below example, the divisor, 1101, is subtracted from the first 4 bits of the dividend, 1001, yielding 100 (the leading 0 of the reminder is dropped).
- The next unused bit from the dividend is then pulled down to make the number of bits in he reminder equal to the total number of bits in the divisor. The next step, therefore, is: 1000-1101, which yields 101, and so on.
- In this process, the divisor always begins with a 1; the divisor is subtracted from portion of the previous dividend / reminder that is equal to it in length. The divisor can only be subtracted from dividend / reminder whose leftmost bit is 1.
- Anytime the leftmost bit of the dividend / reminder is 0, a string of zeros, of the same length as the divisor, replaces the divisor in that step of the process.



- **<u>The CRC checker:</u>** It functions exactly as the CRC generator does. After receiving the data appended with the CRC, it does the same module-2 division. If the reminder is all 0s, the CRC is dropped and the data are accepted; otherwise, the received stream of bits is discarded and data are resent.



- Performance: CRC is a very effective error detection method. If the divisor is chosen according to the previously mentioned rules; can detect all burst errors that affect an odd number of bits.

<u>Checksum</u>

Like the parity check and the CRC, the checksum is based on the concept of redundancy.

- At the **sender**, the **checksum generator** follows these steps:

1) The data unit is divided into **k** sections, each of **n** bits.
2) All sections are added using ones complement to get the sum.
3) The sum is complemented and becomes the checksum.

4) The checksum is sent with the data.

- At the **receiver**, the **checksum checker** follows these steps:

1) The unit into **k** sections each of **n** bits.
2) All sections are added using ones complement to get the sum.
3) The sum is complemented.
4) If the result is zero, the data are accepted, otherwise; they are rejected.

- Example: suppose the following block of 16 bit is to be transmitted using a checksum of 8 bits: 10101001 00111001

- Solution: the numbers are added using ones complement arithmetic:

```
              10101001
              00111001
         Sum  11100010
    Checksum  00011101
```

The pattern sent is: 10101001 00111001 00011101

At the receiver, the receiver will add all the three parts, it will get all 1's, after complementing, the result will be all zeros, which means no error:

```
              10101001
              00111001
              00011101
         Sum  11111111
```

Complement 00000000 means that there is no error.

- Suppose there is a burst error of length that affects 4 bits, as follows:

10101*11*1 *11*111001 00011101

When the receiver adds the three sections, it gets:

```
                    10101111
                    11111001
                    00011101
       Result     1 11000101
     Carry                 1
      Sum           11000110
          Complement 00111001  means the data is corrupted.
```

Error Correction

The mechanisms that we have discussed up to this point detect errors but do not correct them. Error correction can be handled in several ways. The two most common are error correction by retransmission and forward error correction.

- **Error correction by retransmission:** When an error is discovered, the receiver can have the sender retransmit the entire data unit.
- **Forward error correction (FEC):** in this type of correction, a receiver can use an error correcting code, which automatically corrects certain error.

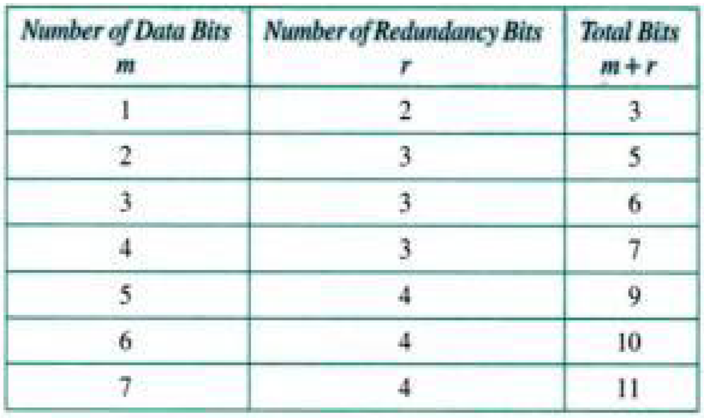
Calculation of Redundancy Bits:

- To calculate the number of redundancy bits r required to correct a given number of data bits m, we must find a relationship between m and r. with m bits of data and r bits of redundancy added to them, the length of the resulting code is equal to r+m.
- If the total number of bits in a transmittable unit is m+r, then, r must be able to indicate at least m+r+1 different states. Of these, one state means no error, and m+r states indicate the location of an error in each of the m+r positions.
- So, m+r+1 states must be discoverable by r bits: and r bits can indicate 2^r different states. Therefore, 2^r must be equal to or greater than m+r+1:

$$2^r >= m+r+1$$

- The value of r can be determined by plugging in the value of m (the original length of data to be transmitted).
- For example, if the value of m is 7 then, the smallest value for r that can satisfy this equation is 4:

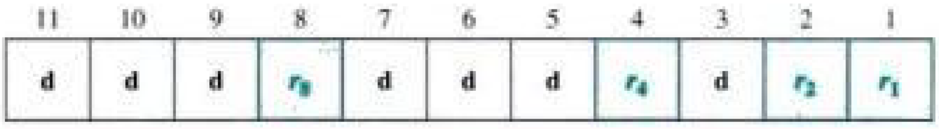
$$2^4 >= 7+4+1$$



Number of Data Bits *m*	*Number of Redundancy Bits* *r*	*Total Bits* *m + r*
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

Hamming Code:

- It can be applied to data units of any length and uses the relationship between data and redundancy bits as discussed above.
- For example, a 7-bit ASCII code requires 4-redundancy bits that can be added to the end of the data unit or interspersed with the original data bits. In figure below, these bits are placed in positions 1, 2, 4 and 8 (the positions in an 11 bit sequence that are powers of 2).
- For clarify more in the examples below, we refer to these bits as r_1, r_2, r_4 and r_8.



- In the Hamming code, each r bit is the parity bit for one combination of data bits, as shown below:

r_1: bits 1, 3, 5, 7, 9, 11 r_2: bits 2, 3, 6, 7, 10, 11

r_4: bits 4, 5, 6, 7 r_8: bits 8, 9, 10, 11

- the arrangement of data bits and the redundancy bits according to hamming code will be as in figure below:
- r_1 will take care of the following bits:

d	**d**	**d**	r_8	**d**	**d**	**d**	r_4	**d**	r_2	r_1

- r_2 will take care of the following bits:

d	**d**	**d**	r_8	**d**	**d**	**d**	r_4	**d**	r_2	r_1

r_4 will take care of the following bits:

d	**d**	**d**	r_8	**d**	**d**	**d**	r_4	**d**	r_2	r_1

r_8 will take care of the following bits:

d	**d**	**d**	r_8	**d**	**d**	**d**	r_4	**d**	r_2	r_1

- **Calculation Of r Values:**

1) The following figure shows a Hamming code implementation for an ASCII character. In the first step, we place each bit of the original character in its appropriate position in the 11-bit unit.
2) Then, we calculate the even parities for various bit combinations.
3) The parity value for each combination is the value of the corresponding **r** bit.

1	**0**	**0**	r_8	**1**	**1**	**0**	r_4	**1**	r_2	r_1
11	**10**	**9**	8	**7**	**6**	**5**	4	**3**	2	1

ling

1	**0**	**0**	r_8	**1**	**1**	**0**	r_4	**1**	r_2	1
11	**10**	**9**	8	**7**	**6**	**5**	4	**3**	2	1

ling

1	**0**	**0**	r_8	**1**	**1**	**0**	r_4	**1**	0	1
11	**10**	**9**	8	**7**	**6**	**5**	4	**3**	2	1

ling											
	1	**0**	**0**	r_8	**1**	**1**	**0**	0	**1**	0	1
	11	10	9	8	7	6	5	4	3	2	1

ling											
	1	**0**	**0**	1	**1**	**1**	**0**	0	**1**	0	1
	11	10	9	8	7	6	5	4	3	2	1

by the time the above transmission is received, the number 7 bit has been changed from 1 to 0. The receiver takes the transmission and recalculates 4 new parity bits, using the same sets of bits used by the sender plus the relevant parity r bit for each set (see figure below). Then it assembles the new parity values into a binary number in order of r position (r_8, r_4, r_2, r_1). In our example, this step gives us the binary number 0111 (7 in decimal), which is the precise location of the b

corrupted

1	**0**	**0**	1	**0**	**1**	**0**	0	**1**	0	1
1	**0**	**0**	1	**0**	**1**	**0**	0	**1**	0	1
1	**0**	**0**	1	**0**	**1**	**0**	0	**1**	0	1
1	**0**	**0**	1	**0**	**1**	**0**	0	**1**	0	1

Means that the bit in position 7 is in error

0 1 1 1

Data communication requires at least two devices working together, one to send and

Flow and Error Control Mechanisms

In this section we introduce three common flow and error control mechanisms: Stop-and-Wait ARQ, Go-Back-*N* ARQ, and Selective-Repeat ARQ. Although these are sometimes referred to as protocols, we prefer the term *mechanisms*.

11.2 STOP-AND-WAIT ARQ

Stop-and-Wait ARQ is the simplest flow and error control mechanism. It has the fol-
lowi

- The sender starts a timer when it sends a frame. If an acknowledgment is not received within an allotted time period, the sender assumes that the frame was lost or damaged and resends it.
- The receiver sends only positive acknowledgment for frames received safe and sound; it is silent about the frames damaged or lost. The acknowledgment number always defines the number of the next expected frame. If frame 0 is received, ACK 1 is sent; if frame 1 is received, ACK 0 is sent.

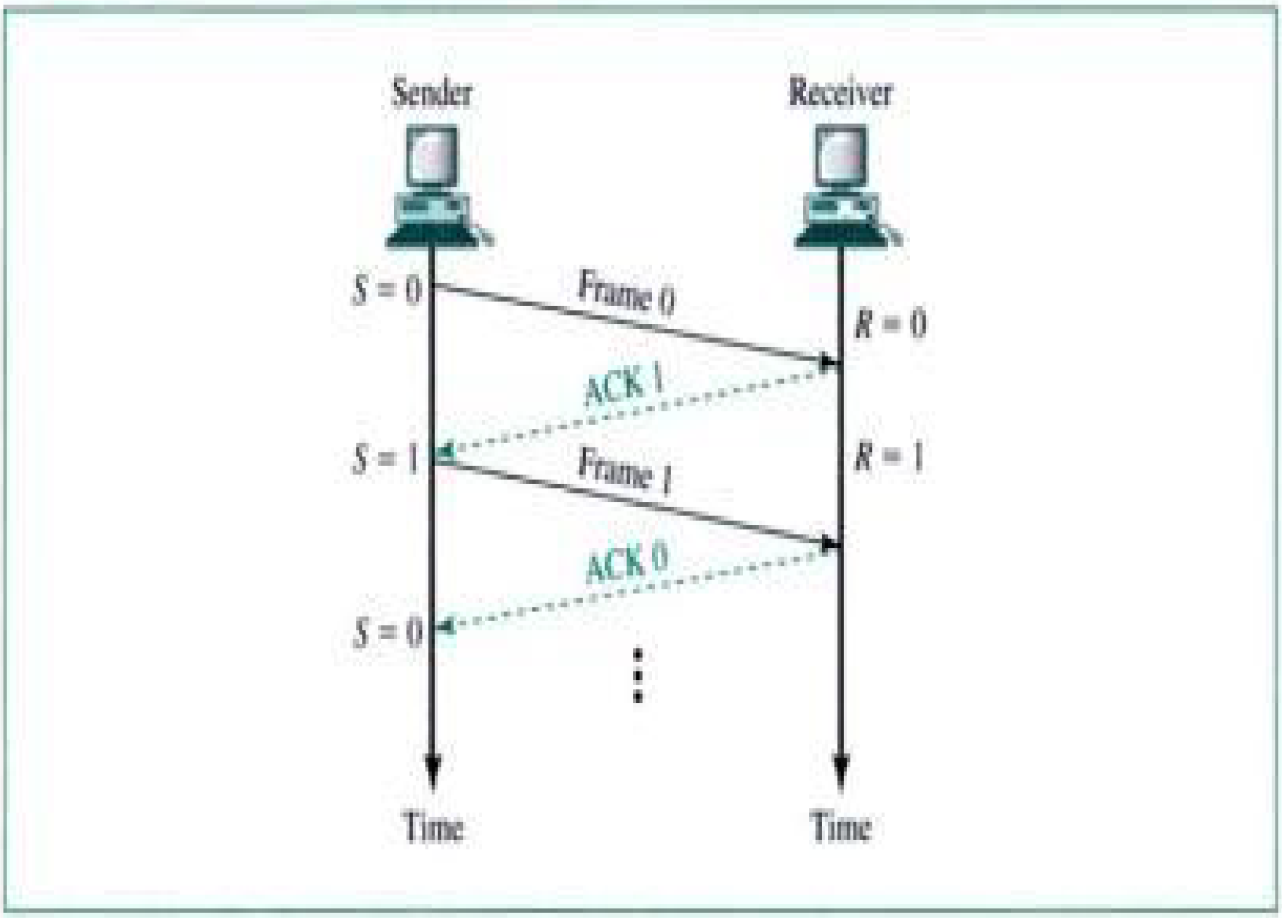
Operation

In the transmission of a frame, we can have four situations: normal operation, the frame is lost, the acknowledgment is lost, or the acknowledgment is delayed.

Normal Operation

In a normal transmission, the sender sends frame 0 and waits to receive ACK 1. When
rec ACK 1 is received, it sends frame 1 and then waits to receive ACK 0, and so on. The
rec ACK must be received before the timer set for each frame expires. Figure 11.1 shows
successful frame transmissions.

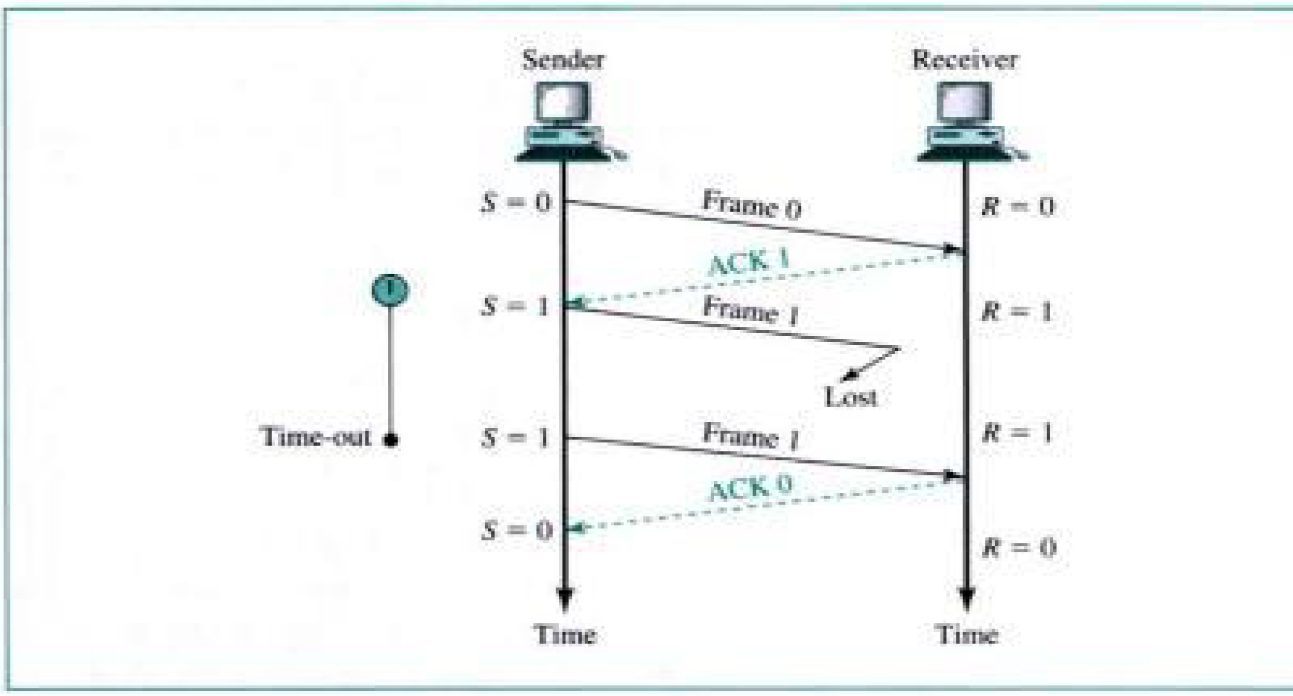
Figure 11.1 *Normal operation*



Lost or Damaged Frame

A lost or damaged frame is handled in the same way by the receiver; when the receiver receives a damaged frame, it discards it, which essentially means the frame is lost. The receiver remains silent about a lost frame and keeps its value of *R*. For example, in Figure 11.2, the sender transmits frame 1, but it is lost. The receiver does nothing, retaining the value of *R* (1). After the timer at the sender site expires, another copy of frame 1 is sent.

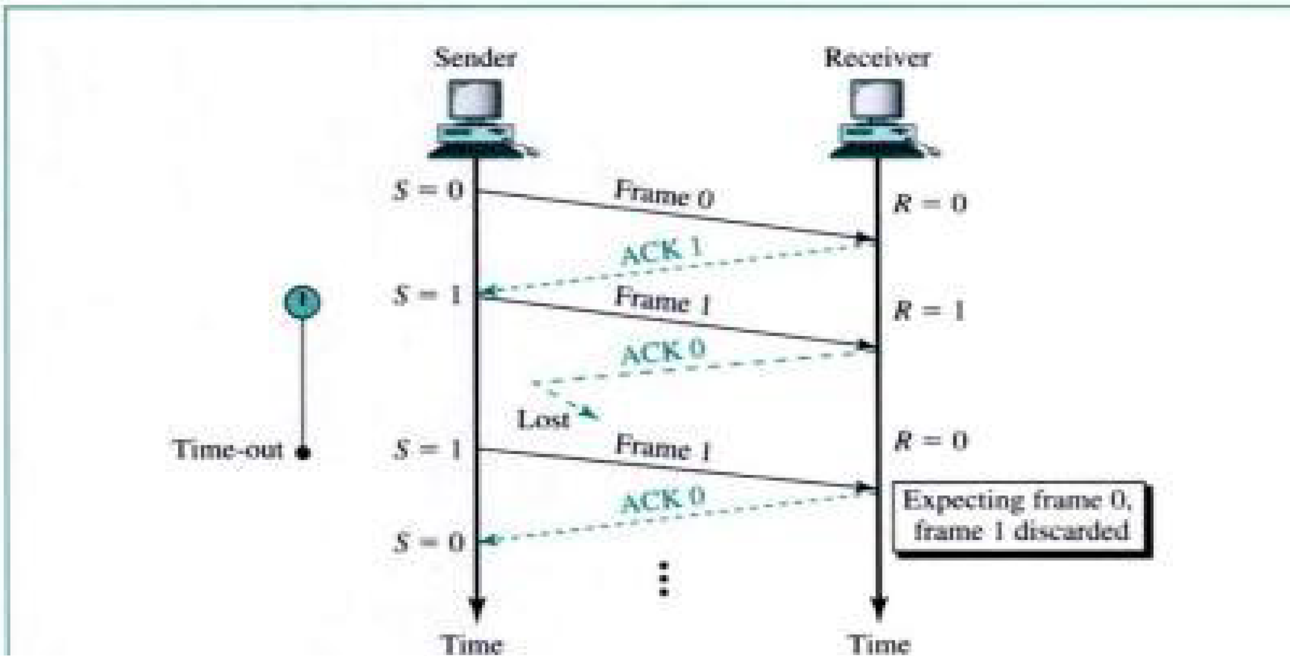
Figure 11.2 *Stop-and-Wait ARQ, lost frame*



Lost Acknowledgment

A lost or damaged acknowledgment is handled in the same way by the sender; if th sender receives a damaged acknowledgment, it discards it. Figure 11.3 shows a lo ACK 0. The waiting sender does not know if frame 1 has been received. When th timer for frame 1 expires, the sender retransmits frame 1. Note that the receiver ha already received frame 1 and is expecting to receive frame 0 ($R = 0$). Therefore, silently discards the second copy of frame 1.

Figure 11.3 *Stop-and-Wait ARQ, lost ACK frame*



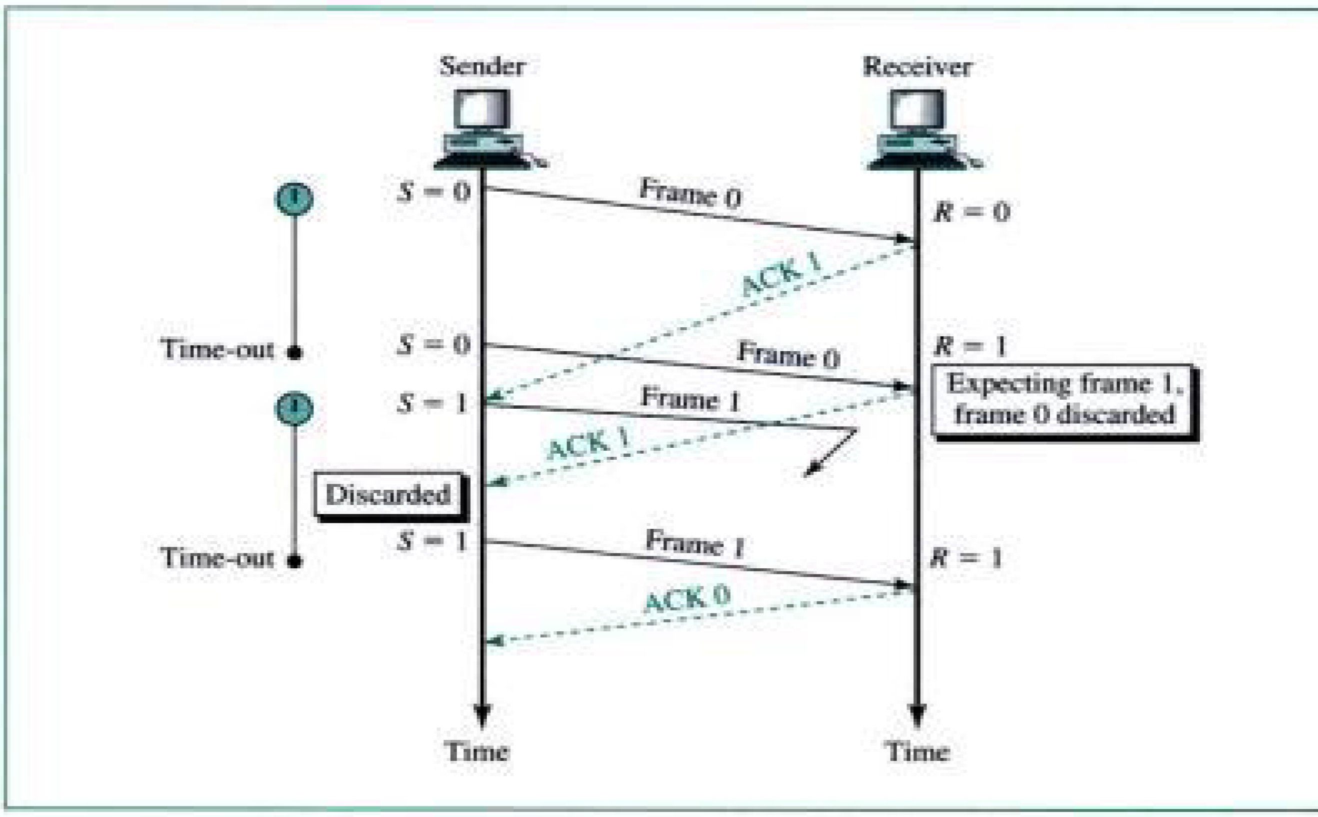
The reader may have discovered through this example the need to number fram
If the frames were not numbered, the receiver, thinking that frame 1 is a new frame (n
a duplicate), keeps it.

In Stop-and-Wait ARQ, numbering frames prevents the retaining of duplicate frames.

Delayed Acknowledgment

Another problem that may occur is delayed acknowledgment. An acknowledgment c
be delayed at the receiver or by some problem with the link. Figure 11.4 sho
the delay of ACK 1; it is received after the timer for frame 0 has already expired. T
sender has already retransmitted a copy of frame 0. However, the value of *R*
the receiver site is still 1, which means that the receiver expects to see frame 1. T
receiver, therefore, discards the duplicate frame 0.

Figure 11.4 *Stop-and-Wait ARQ, delayed ACK*



The sender has now received two ACKs, one that was delayed and one that w
sent after the duplicate frame 0 arrived. The second ACK 1 is discarded.

To understand why we need to number the acknowledgments, let us examine Fi
ure 11.4 again. After the delayed ACK 1 reaches the sender, frame 1 is sent. Howev
frame 1 is lost and never reaches the receiver. The sender then receives an ACK 1 f
the duplicate frame sent. If the ACKs were not numbered, the sender would interp

Numbered acknowledgments are needed if an acknowledgment is delayed and the next frame is lost.

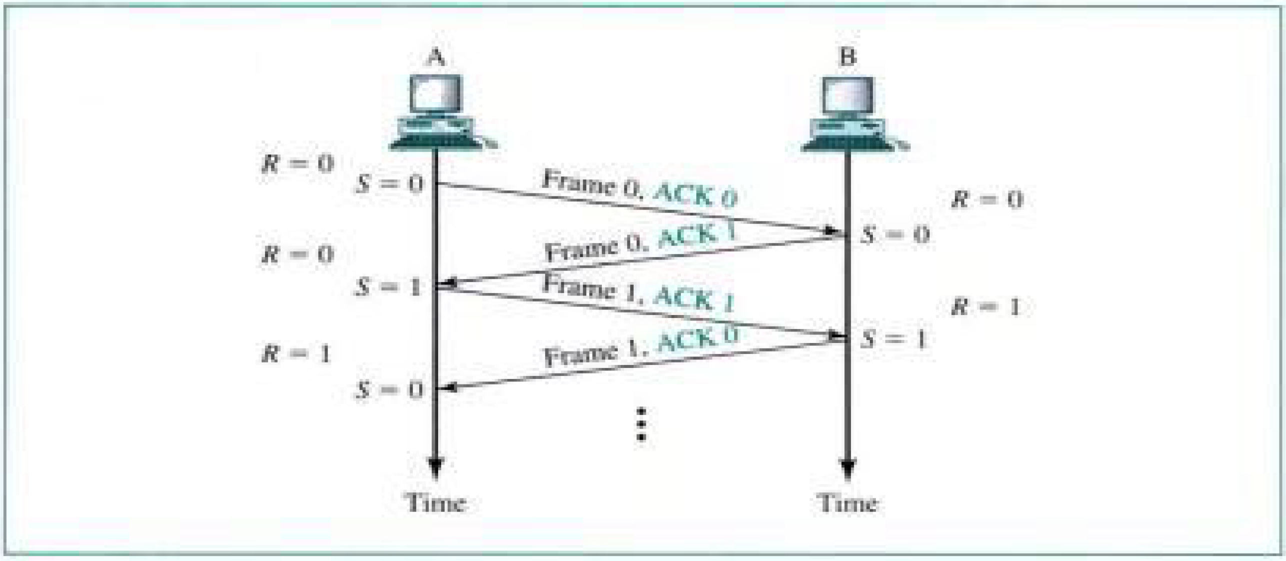
Bidirectional Transmission

The stop-and-wait mechanism we have discussed is unidirectional. However, we can have bidirectional transmission if the two parties have two separate channels for full-duplex transmission or share the same channel for half-duplex transmission. In this case, each party needs both *S* and *R* variables to track frames sent and expected.

Piggybacking

Piggybacking is a method to combine a data frame with an acknowledgment. For example, in Figure 11.5, Stations A and B both have data to send. Instead of sending separate data and ACK frames, station A sends a data frame that includes an ACK. Station B behaves in a similar manner.

Figure 11.5 *Piggybacking*



Piggybacking can save bandwidth because the overhead from a data frame and an ACK frame (addressees, CRC, etc.,) can be combined into just one frame.

11.3 GO-BACK-*N* ARQ

In Stop-and-Wait ARQ, at any point in time for a sender, there is only one frame, the outstanding frame, that is sent and waiting to be acknowledged. This is not a good use of the transmission medium. To improve the efficiency, multiple frames should be in transition while waiting for acknowledgment. In other words, we need to let more than

one frame be outstanding. Two protocols use this concept: **Go-Back-*N* ARQ** and **Selective Repeat ARQ.** We discuss the first in this section and the second in Section 11.4.

In Go-Back-*N* ARQ, we can send up to *W* frames before worrying about acknowledgments; we keep a copy of these frames until the acknowledgments arrive. This procedure requires additional features to be added to Stop-and-Wait ARQ.

Sequence Numbers

Frames from a sending station are numbered sequentially. However, because we need to include the sequence number of each frame in the header, we need to set a limit. If the header of the frame allows *m* bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$. For example, if *m* is 3, the only sequence numbers are 0 through 7 inclusive. However, we can repeat the sequence. So the sequence numbers are

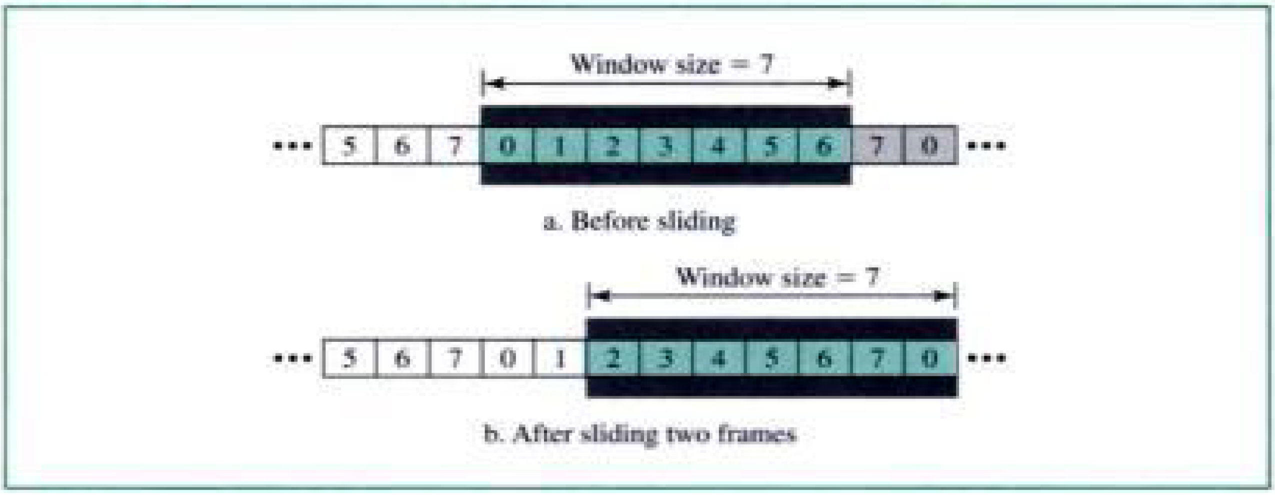
0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, . . .

Sender Sliding Window

At the sender site, to hold the outstanding frames until they are acknowledged, we use the concept of a window. We imagine that all frames are stored in a buffer. The outstanding frames are enclosed in a window. The frames to the left of the window are those that have already been acknowledged and can be purged; those to the right of the window cannot be sent until the window slides over them. The size of the window is at most $2^m - 1$ for reasons that we discuss later.

The size of the window in this protocol is fixed, although we can have a variable-size window in other protocols such as TCP (see Chapter 22). The window slides to include new unsent frames when the correct acknowledgments are received. The window is a **sliding window.** For example, in Figure 11.6*a*, frames 0 through 6 have been sent. In part *b*, the window slides two frames over because an acknowledgment was received for frames 0 and 1.

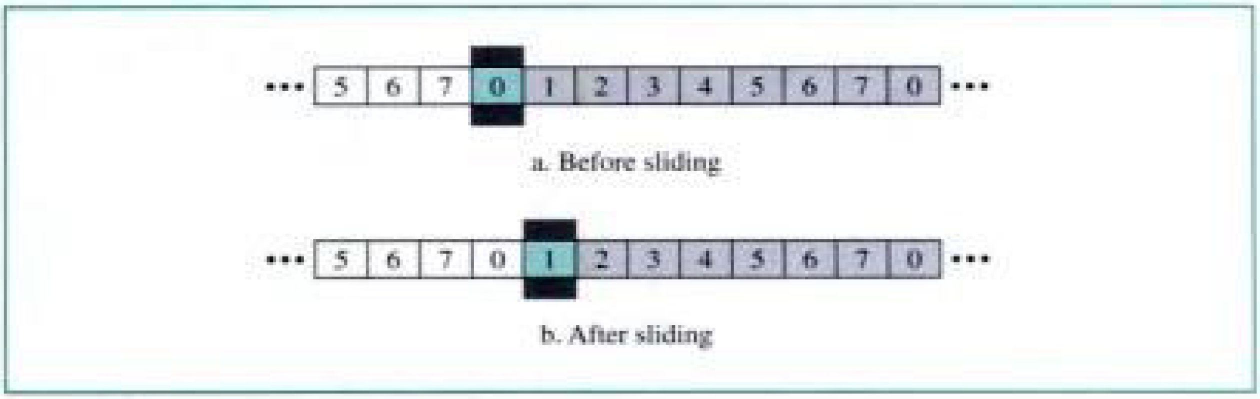
Figure 11.6 *Sender sliding window*



Receiver Sliding Window

The size of the window at the receiver site in this protocol is always 1. The receiver is always looking for a specific frame to arrive in a specific order. Any frame arriving out of order is discarded and needs to be resent. The receiver window also slides as shown in Figure 11.7. In part *a* the receiver is waiting for frame 0. When that arrives, the window slides over.

Figure 11.7 *Receiver sliding window*

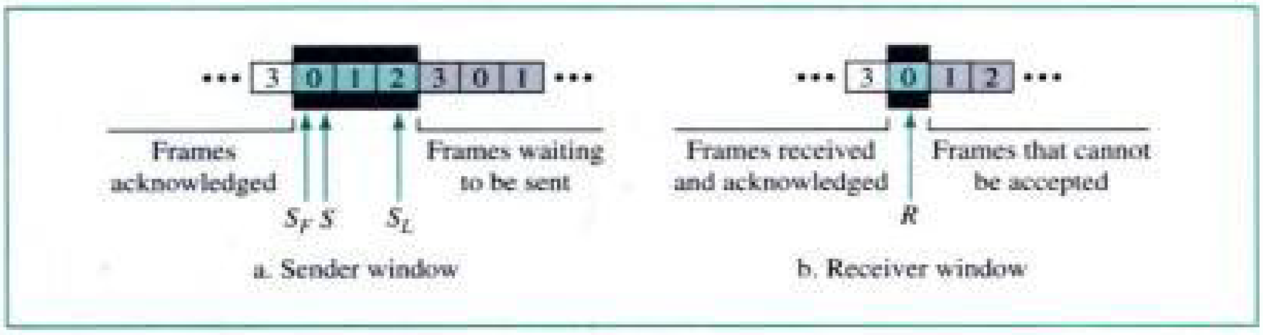


Control Variables

The sender has three variables, S, S_F, and S_L. The S variable holds the sequence number of the recently sent frame; S_F holds the sequence number of the first frame in the window; and S_L holds the sequence number of the last frame in the window. The size of the window is W, where $W = S_L - S_F + 1$.

The receiver only has one variable, R, that holds the sequence number of the frame it expects to receive. If the sequence number of the received frame is the same as the value of R, the frame is accepted; if not, it is rejected. Figure 11.8 shows the sender and receiver window with their control variables.

Figure 11.8 *Control variables*



Timers

The sender sets a timer for each frame sent. The receiver has no timers.