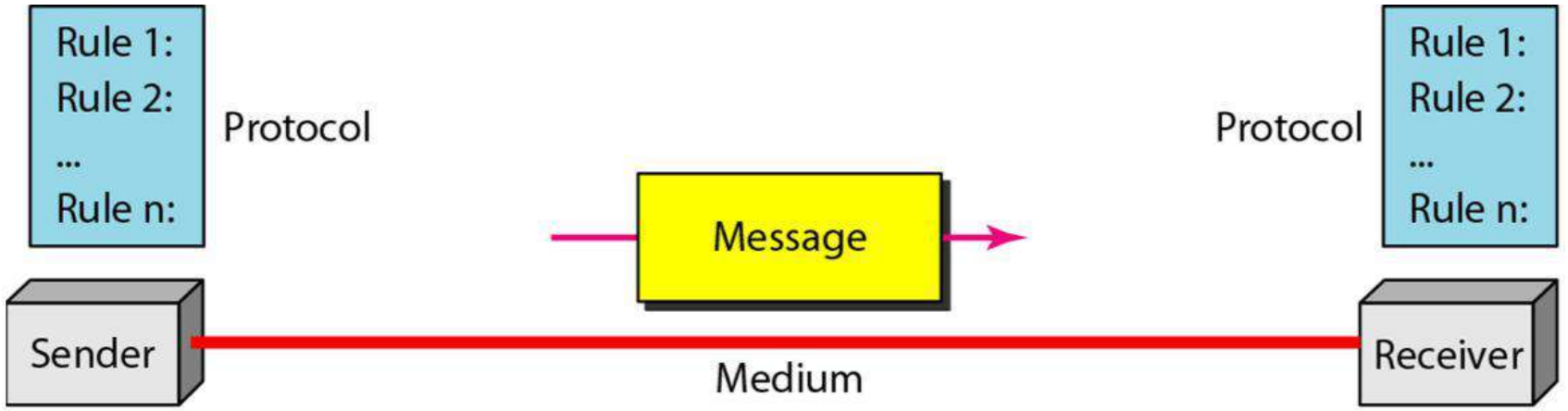
Network Management Branch

Network Management course1

Assist.Prof.Dr. Ekhlas Khalaf

Data Communication System

- Message: The message is the information (data) to be communicated.
- Sender: The sender is the device that sends the data message.
- Receiver: The receiver is the device that receives the data message.
- Transmission medium: The transmission medium is the physical path by which a message travels from sender to receiver.
- Protocol: A protocol is a set of rules that govern data communications.

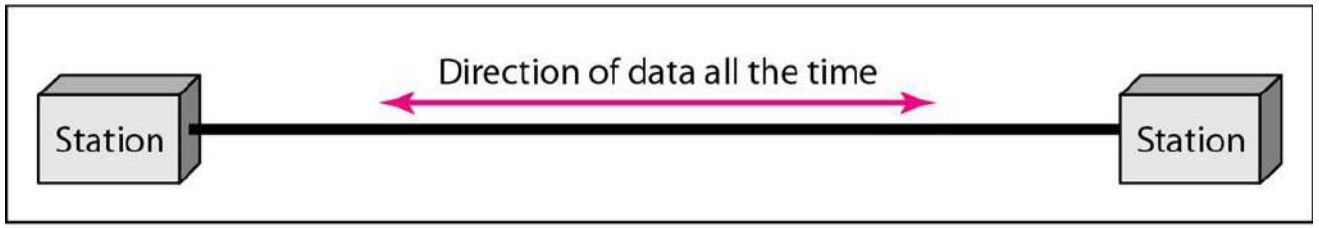


Data Flow

- Communication between two devices can be

a. Simplex

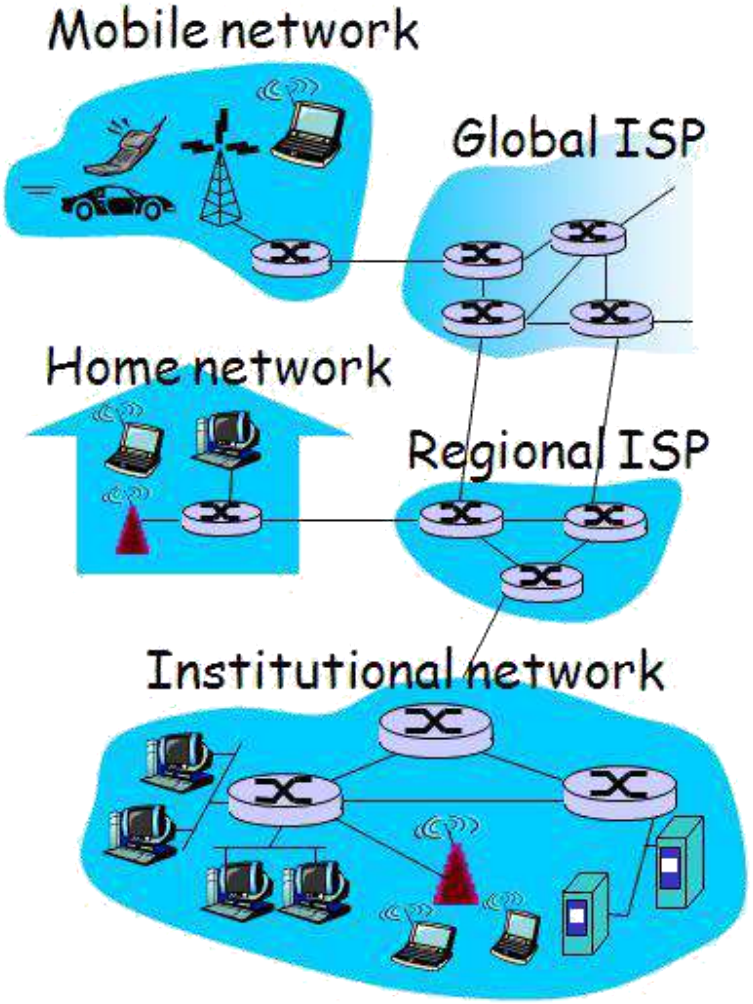
b. Half-duplex



c. Full-duplex

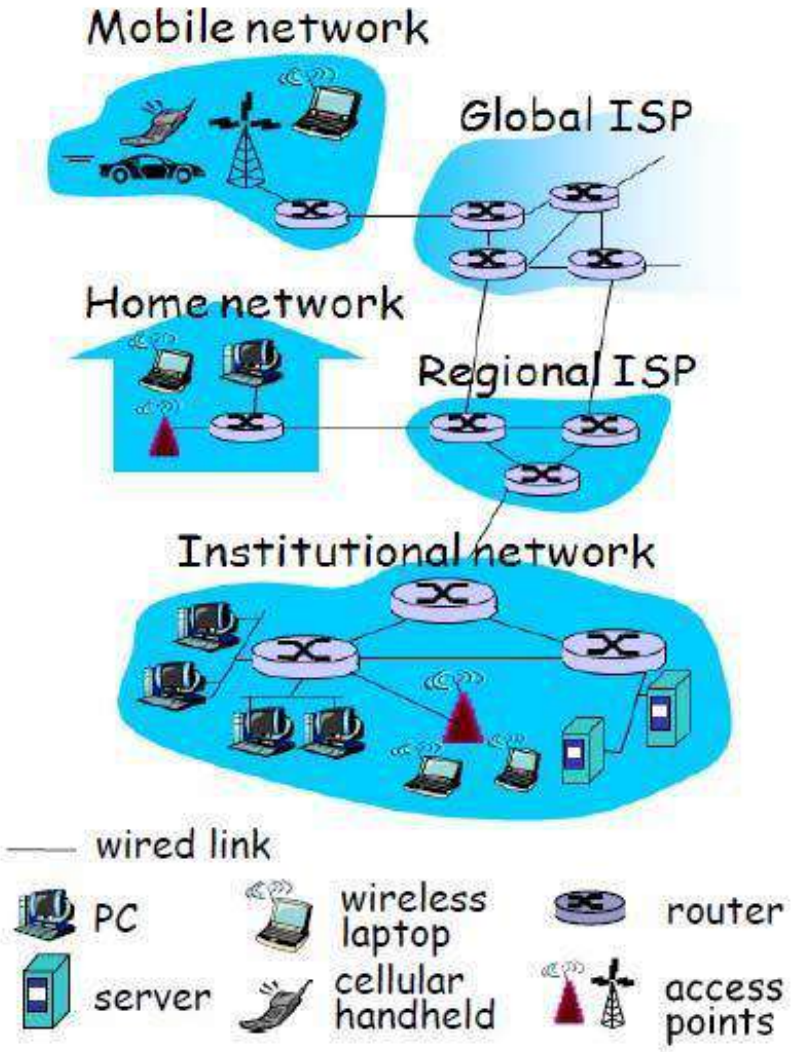
What's the Internet?

- ***Hosts = end systems***
  - millions of connected computing devices
  - running *network apps*
- ***Communication links***
  - fiber, copper, radio, satellite
  - transmission rate = ***bandwidth***
- ***Routers***
  - forward packets (chunks of data)



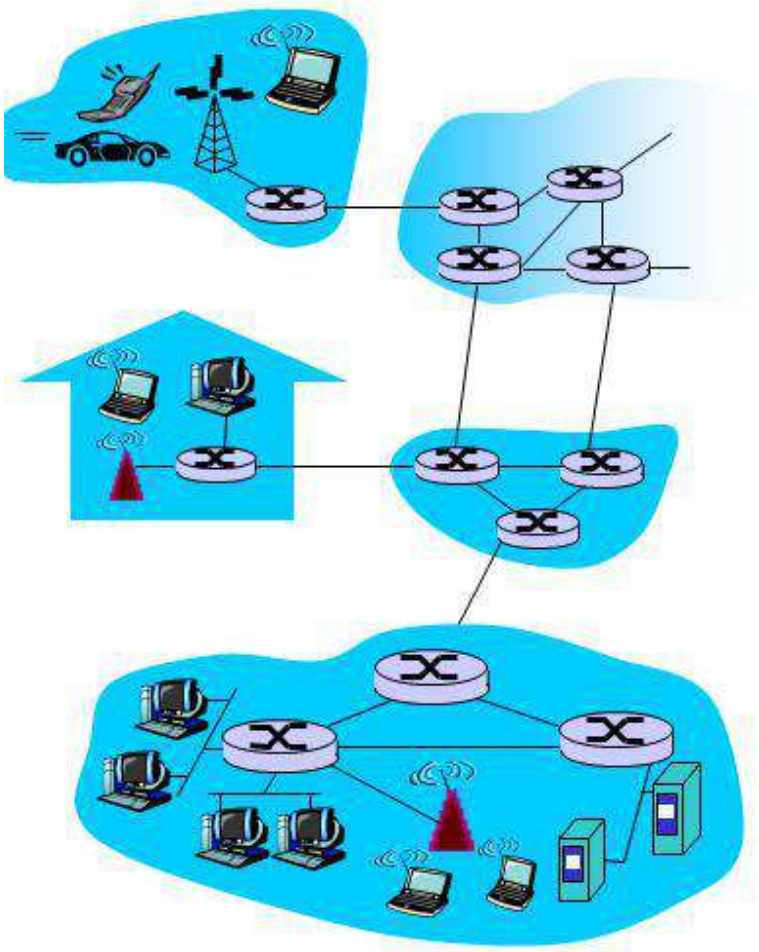
What's the Internet?

- ***Protocols***
  - control sending, receiving of msgs
  - e.g., TCP, IP, HTTP, FTP, PPP
- ***Internet***
  - *"network of networks"*
  - loosely hierarchical
  - public Internet vs. private intranet
- **Internet standards**
  - RFC: Request for Comments
  - IETF: Internet Engineering Task Force



A Closer Look at Network Structure

- **Network edge**
  - applications and hosts
- **Network core**
  - interconnected routers
  - network of networks
- **Access networks, physical media**
  - wired, wireless communication links



Network Edge: Connection-Oriented Service

- Goal
  - data transfer between end systems
- **TCP - Transmission Control Protocol**
  - Internet's connection-oriented service
  - TCP service [RFC 793]
- **Reliable, in-order byte-stream data transfer**
  - loss: acknowledgements and retransmissions
- ***Flow control***
  - sender won't overwhelm receiver
- Congestion control
  - senders "slow down sending rate" when network congested

Network Edge: Connectionless Service

- **Goal**
  - data transfer between end systems
  - same as before!
- **UDP - User Datagram Protocol [RFC 768]**
  - connectionless
  - unreliable data transfer
  - no flow control
  - no congestion control
- **App's using UDP**
  - streaming media, teleconferencing, DNS, Internet telephony

The Network Core

- The fundamental question:
  How is data transferred through net?

- **circuit switching**
  - dedicated circuit per call (telephone net)

- **packet-switching**
  - data sent through net in discrete “chunks”

- **cell switching**
  - data is segmented into small cells of fixed size.

Protocol “Layers”

- Networks are complex!
- Many “pieces”

  hosts

  routers

  links of various media

  applications

  protocols

  hardware

  software

- Q: Is there any hope of organizing structure of network?

Why Layering?

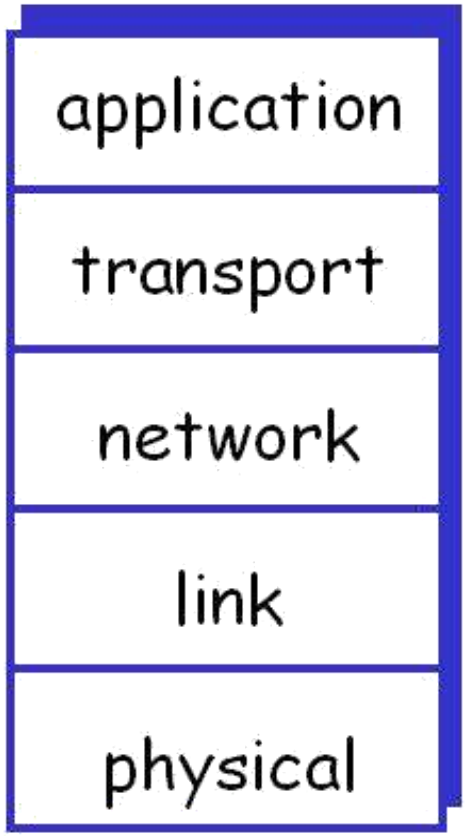
- Dealing with complex systems
- Explicit structure allows identification, relationship of complex system's Pieces

  layered reference model for discussion
- Modularization eases maintenance, updating of system

  change of implementation of layer's service transparent to rest of system

Internet Protocol Stack

- **Application**
  supporting network applications
  FTP, SMTP, HTTP
- **Transport**
  process-process data transfer
  TCP, UDP
- **Network**
  routing of datagrams from src to
  dest IP, routing protocols
- **Link**
  data transfer between neighboring network
  elements PPP, Ethernet
- **Physical**
  bits “on the wire”



Literature:

- Books
  - Behrouz A. Forouzan: Data Communications and Networking, 2007
  - Administration CISCO QoS in IP Networks
  - Alexander Clemm, Network Management Fundamentals, Ph.D. , 2007 Cisco Systems, Inc.

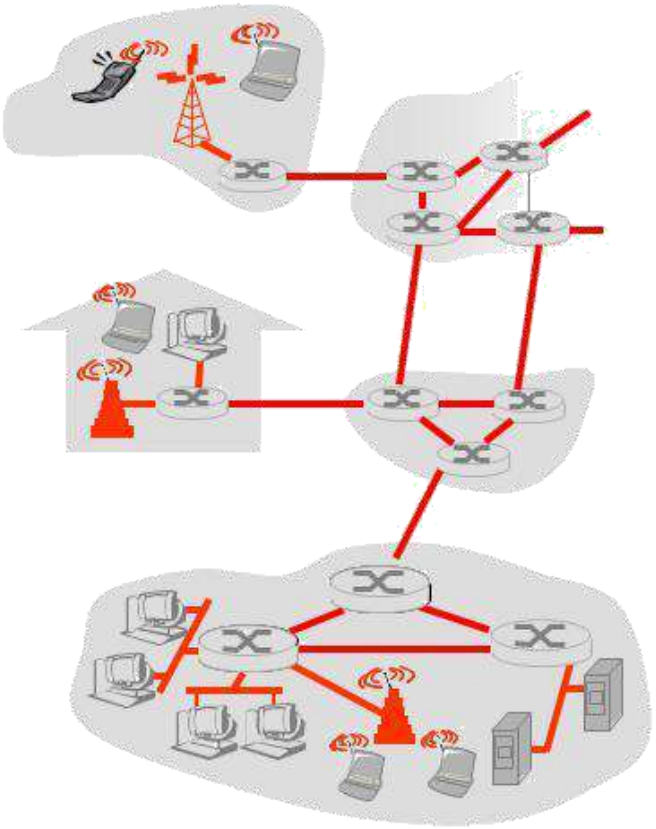
- Many articles → Google scholar

Network Management

2. Data Link Control

Link Layer: Introduction

■ Some terminology



- hosts and routers are *nodes*

communication channels that connect adjacent nodes along communication path are *links*

- wired links
- wireless links
- LANs

layer-2 packet is a *frame*

- encapsulates datagram

Data-Link Layer

Has responsibility of transferring datagram from one node to adjacent node over a link

Link Layer: Context

- Datagram transferred by different link protocols over different links
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- Each link protocol provides different services
  - e.g., may or may not provide rdt over link

Transportation Analogy

Trip from Princeton to Lausanne

- Princeton to JFK by limo, JFK to Geneva by plane, Geneva to Lausanne by train
- tourist = datagram
- transport segment = communication link
- transportation mode = link layer protocol
- travel agent = routing algorithm

Link Layer: Services (1)

- Framing, link access

  encapsulate datagram into frame, adding

  header/trailer channel access if shared medium

  “MAC” addresses used in frame headers to identify

  source/destination different from IP address!

- Reliable delivery between adjacent nodes

  seldom used on low bit error link (fiber, some twisted pair)

  wireless links with typically high error rates

Link Layer: Services (2)

- *Flow Control*
  pacing between adjacent sending and receiving nodes

- *Error Detection*
  errors caused by signal attenuation, noise
  receiver detects presence of errors
  signals sender for retransmission or drops frame

- Error Correction:
  receiver identifies and *corrects* bit error(s) without resorting to retransmission

- *Half-duplex* and *full-duplex*
  with half duplex, nodes at both ends of link can transmit, but not at the same time.

Link Layer: Implementation

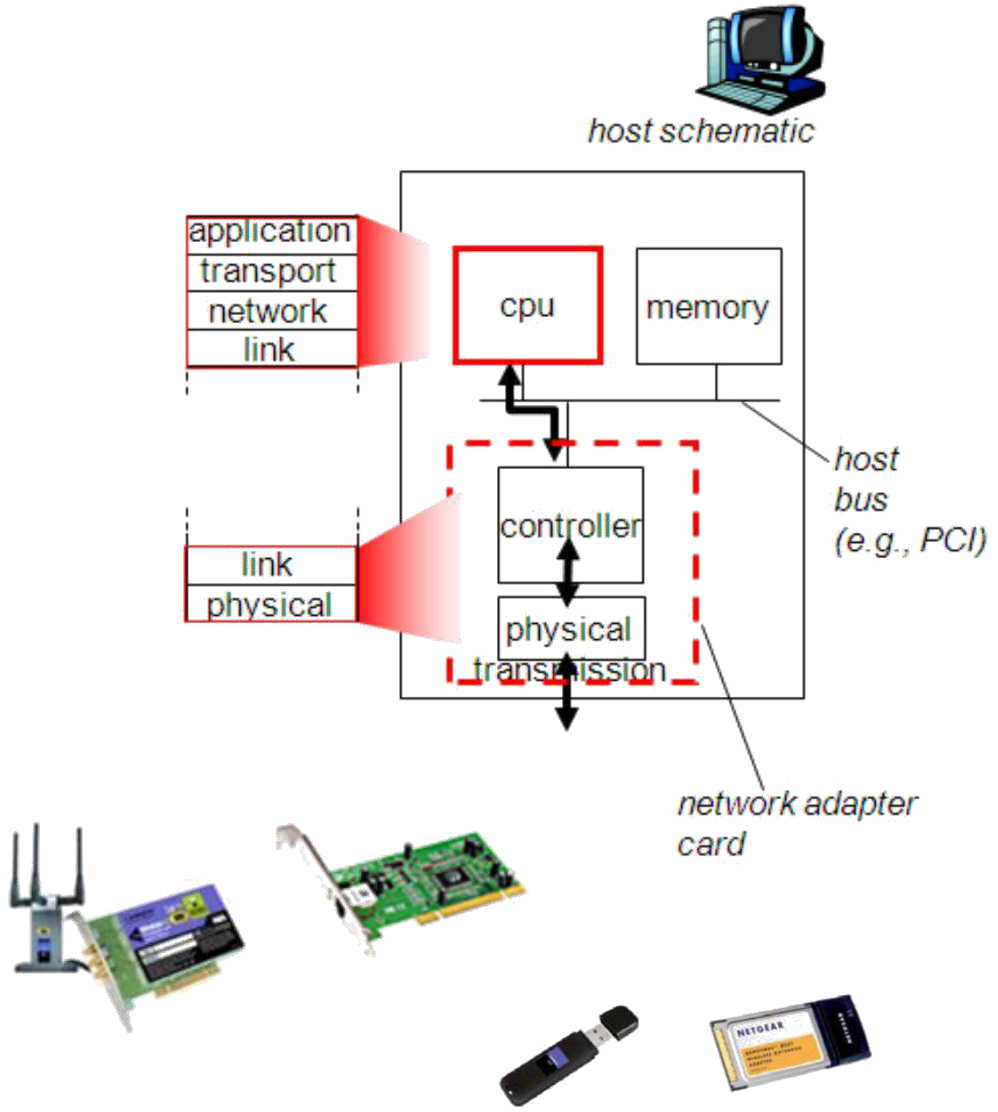
- Implemented

  in each and every host

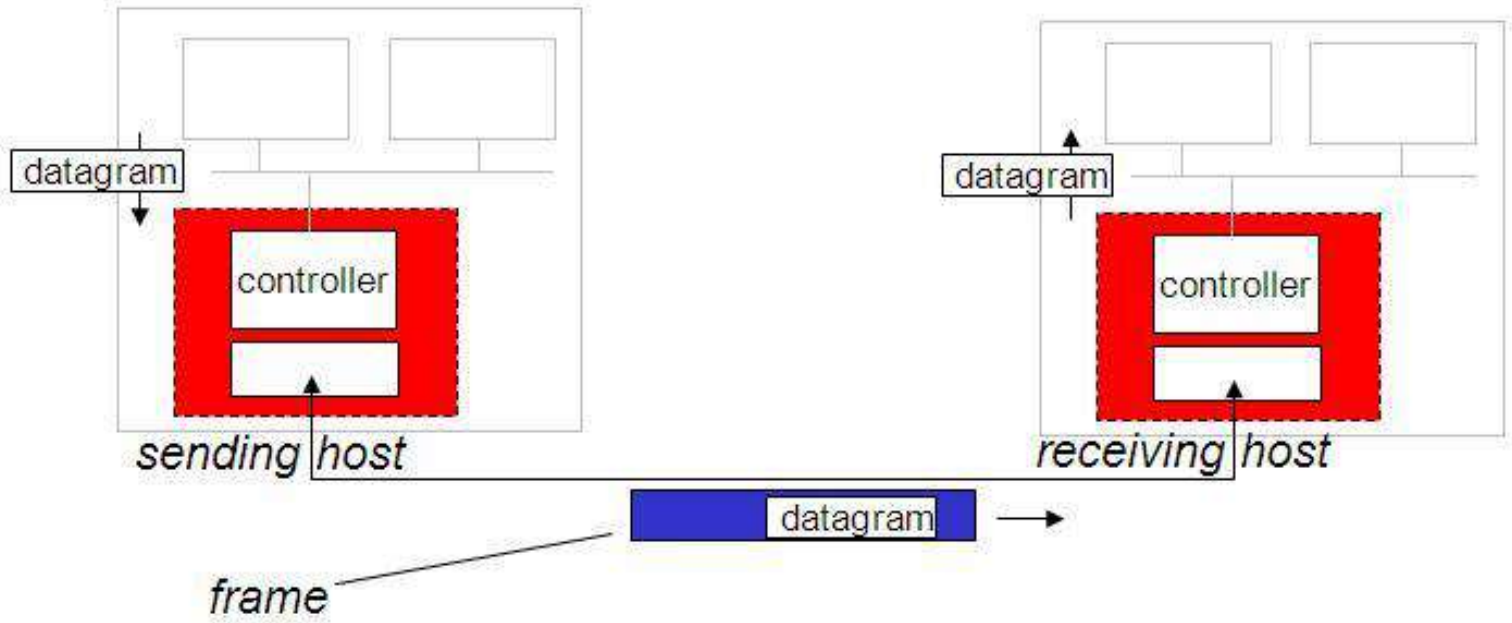
  in semi-autonomous “adaptor“
  - aka *network interface card (NIC)*
  - e.g. Ethernet card, PCMCIA card, 802.11 card
  - link and physical layer

  attaches into host‘s system

  buses combination of hardware, software, firmware



Adaptors Communicating



Sending side

encapsulates datagram in a frame

adds error checking bits, etc.

Receiving side

looks for errors, rdt, flow control, etc.

extracts datagram, passes to receiving node

Data Link Layer

- Data link control, deals with the design and procedures for communication between two adjacent nodes.
- Media access control, how to share the link.

- Data link control functions include framing, flow and error control, and software implemented protocols that provide smooth and reliable transmission of frames between nodes.

- The IEEE has subdivided the data link layer into two sublayers: logical link control (LLC) and media access control (MAC).

- Q: what do we need to implement data link control communication between two adjacent nodes ?

2.1 FRAMING

- Data transmission in the physical layer means moving bits in the form of a signal from the source to the destination. The physical layer provides bit synchronization to ensure that the sender and receiver use the same bit durations and timing.

- The data link layer, on the other hand, needs to pack bits into frames, so that each frame is distinguishable from another.

- Framing in the data link layer separates a message from one source to a destination, or from other messages to other destinations, by adding a sender address and a destination address. The destination address defines where the packet is to go; the sender address helps the recipient acknowledge the receipt.

- Q : Could the whole message be packed in one frame ?

Fixed-Size Framing

Frames can be of fixed or variable size. In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter. An example of this type of framing is the ATM (Asynchronous Transfer Mode) wide-area network, which uses frames of fixed size called cells.

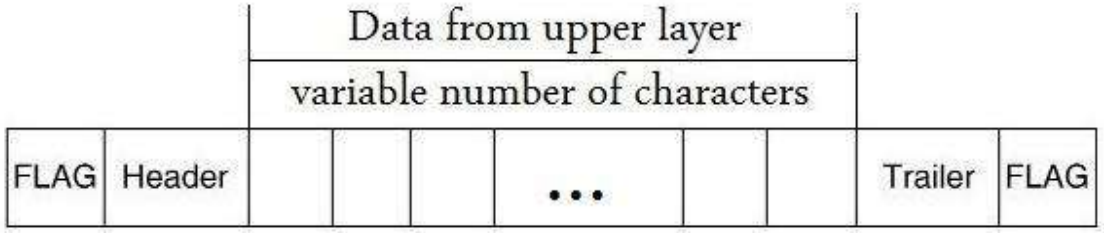
Variable-Size Framing

In variable-size framing, we need a way to define the end of the frame and the the beginning of the next. Historically, two approaches were used for this purpose: a character-oriented approach and a bit-oriented approach.

Character-Oriented Protocols

- In a character-oriented protocol, data to be carded are 8 -bit characters from a coding system. The header, which normally carries the source and destination addresses and other control information, and the trailer,

which carries error detection or error correction redundant bits, are also are also multiples of 8 bits. To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame.



□ Character-oriented framing was popular when only text was exchanged by the data link layers. The flag could be selected to be any character not used for text communication. Now, however, we send other types of information such as graphs, audio, and video. Any pattern used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame.

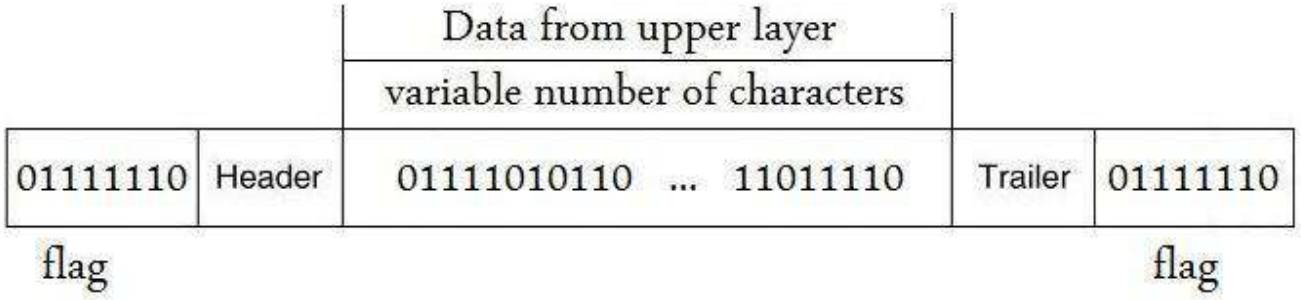
To fix this problem, a byte-stuffing strategy was added to character oriented framing. In byte stuffing, a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. This byte is usually called the escape character (ESC), which has a predefined bit pattern.

- Q : What happens if the text contains one or more escape characters followed by a flag?

A: The receiver removes the escape character, but keeps the flag, which is incorrectly interpreted as the end of the frame. To solve this problem, the escape characters that are part of the text must also be marked by another escape character. In other words, if the escape character is part of the text, an extra one is added to show that the second one is part of the text. Character-oriented protocols present another problem in data communications. The universal coding systems in use today have 16-bit and 32-bit characters that conflict with 8-bit characters.

■ Bit-Oriented Protocols

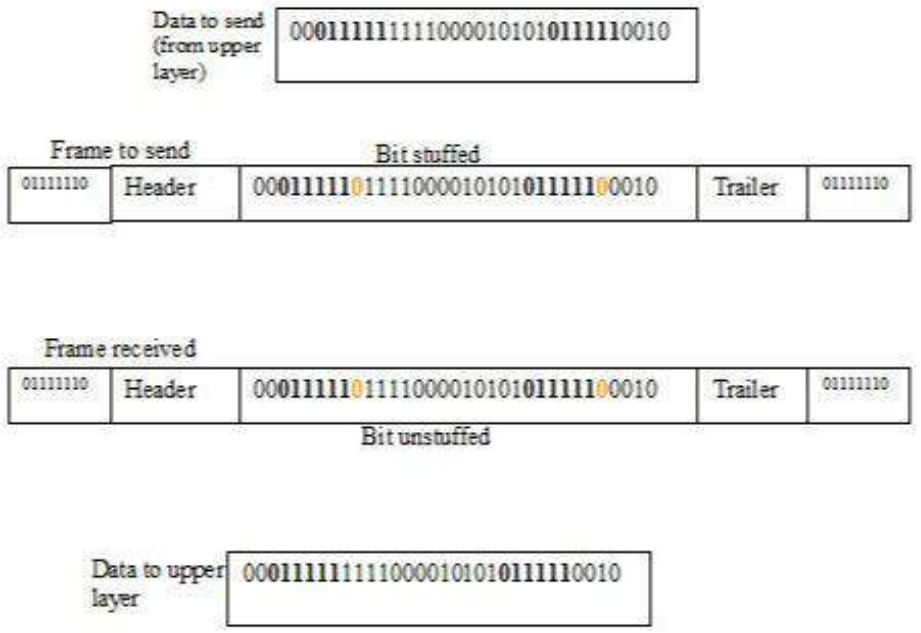
□ In a bit-oriented protocol, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on. However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other. Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame.



This flag can create the same type of problem we saw in the byte -oriented protocols. That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame. We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from looking like a flag.

- Bit stuffing is the process of adding one extra 0 whenever five consecutive is follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.

This means that if the flag like pattern 01111110 appears in the data, it will change to 011111010 (stuffed) and is not mistaken as a flag by the receiver. The real flag 01111110 is not stuffed by the sender and is recognized by the receiver.



2.2 FLOW AND ERROR CONTROL

Data communication requires at least two devices working together, one to send and the other to receive. Even such a basic arrangement requires a great deal of coordination for a clear exchange to occur. The most important responsibilities of the data link layer are flow control and error control.

■ Flow Control

□ Flow control coordinates the amount of data that can be sent before receiving an acknowledgment and is one of the most important duties of the data link layer. In most protocols, flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver. The flow of data must not be allowed to overwhelm the receiver. Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data. The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stop temporarily. The rate of such processing is often slower than the rate of

transmission. For this reason, each receiving device has a block of memory, called a buffer, reserved for storing incoming data until they are processed. If the buffer begins to fill up, the receiver must be able to tell the sender to halt transmission until it is once again able to receive.

- Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.

Error Control

- Error control is both error detection and error correction. It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender. In the data link layer, the term error control refers primarily to methods of error detection and retransmission.

- Error control in the data link layer is often implemented simply: Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ).

■ Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.

Error Detection and Correction

Many applications require a correct data-transmission i.e. data reaching a destination is the same than the data transmitted.

- Problem:

Channels are not an ideal medium

- noise/interferences on the channel
- Losses on purpose (e.g. dropping packets in case of congestion)

- Errors occur because of
  - attenuation: Loss of energy to overcome the resistance of the medium.
  - distortion: Signal changes its shape.
  - noise: Addition of unwanted signals.

Error types

- data loss
- data alternation
- data duplication, miss insertion or wrong delivery
- failure of components (usually not considered for multimedia systems)
- single-bit error: changes of one bit

• Happens in: Parallel transmission
• Example: 10110101 ⟶ 10110111

- burst error: changes of more than one bit

• Happens in: Serial transmission
• Reason: Noise duration is more than one bit data duration
• Example: 10110101 ⟶ 10100111

■ Q: How do we detect this?

■ Mechanisms

Error detection: Detection of one or more corrupted bits in a bit string

- Parity check (one or two dimensional)
- Checksum
- Cyclic Redundancy Check (CRC)

- not all conceivable errors can be detected!

Error handling/correction: Detection and correction of corrupted bits

- Means: retransmission of data (e.g. in TCP), employing coding theory, Hamming code

- Not all conceivable errors can be corrected!
- Redundant bits are added to detect and correct errors

Error Detection Mechanisms

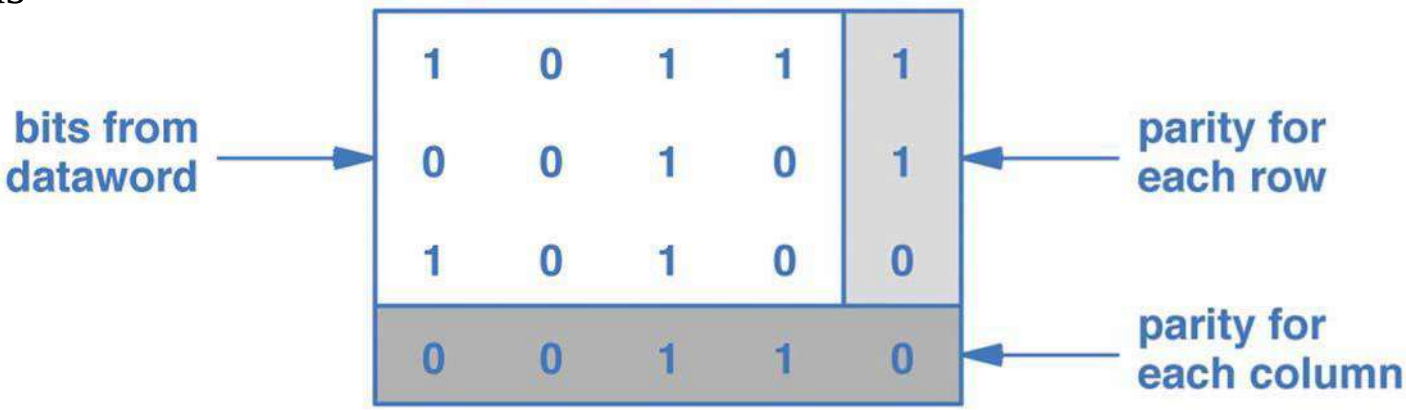
■ Parity Check

- Simplest error detection scheme
- Used for a single bit error detection
- A parity bit is added at the end of a block of data

• Example: ASCII G: 111001 ⟶ 1110010 (even parity)
⟶ 1110011 (odd parity)

Two dimensional parity check

- Improved over simple parity check
- Data are arranged in a table and parity bits are calculated for both row and columns
- Example:



■ Checksum

Several protocols use checksum algorithms for example: TCP/IP, ICMP

sender side

- message divided into m bit words
- all words added using one's complement arithmetic
- sum complemented and becomes the checksum
- checksum is sent with the data

receiver side

- Message including checksum is divided into m bit words
- All words added using one's complement arithmetic
- Sum is complemented and becomes the new checksum
- If value of checksum is 0, then message is accepted, otherwise, rejected.

- Example: Data 10110110 11010101 01011011
- Sender side:

```
1 0 1 1 0 1 1 0 ←
1 1 0 1 0 1 0 1 ←
---------------
1 0 0 0 1 0 1 1
        1
---------------
1 0 0 0 1 1 0 0
0 1 0 1 1 0 1 1 ←
---------------
1 1 1 0 0 1 1 1
---------------
0 0 0 1 1 0 0 0
```

Message sent:

10110110 11010101
01011011 00011000

- Receiver side

```
1 0 1 1 0 1 1 0 ←
1 1 0 1 0 1 0 1 ←
---------------
1 0 0 0 1 0 1 1
        1
---------------
1 0 0 0 1 1 0 0
0 1 0 1 1 0 1 1 ←
---------------
1 1 1 0 0 1 1 1
0 0 0 1 1 0 0 0 ←
---------------
1 1 1 1 1 1 1 1
---------------
0 0 0 0 0 0 0 0 ← No Error
```

Performance of checksum algorithms

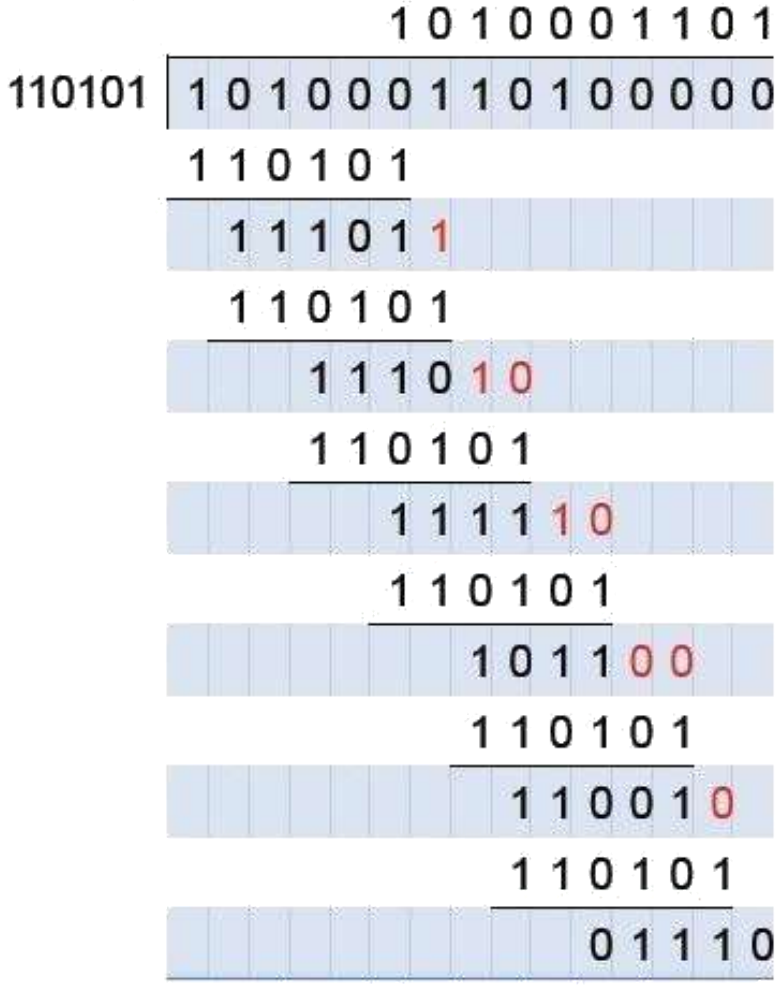
- can not detect error
- When the value of one word is incremented and the value of another word is decremented by the same amount
- Nowadays, checksum is usually replaced with Cyclic Redundancy Check (CRC)

■ Cyclic Redundancy Check (CRC)

most common and most powerful error detecting
code usually used in data link layer
can detect 100% single and double bit errors
(Message bits (k) + transmitter generated bits (n)) /predetermined
number receiver divides the incoming message with that predetermined
number zero reminder means no error

Example:

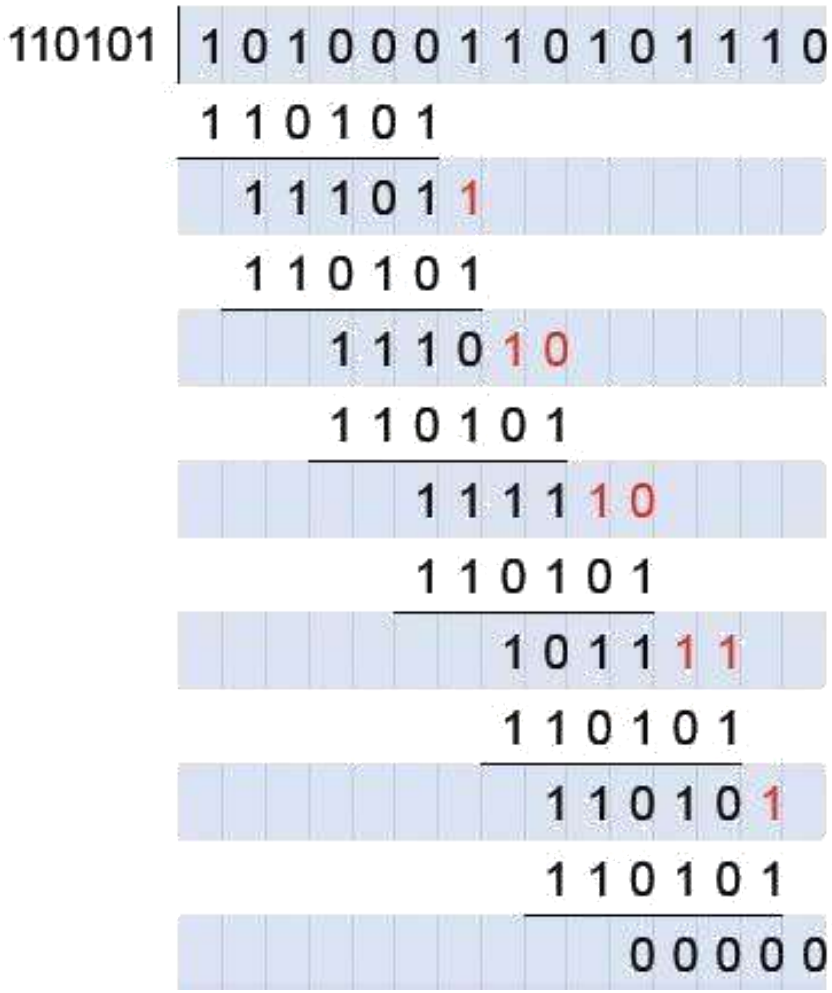
- Given, Data = 1010001101, Divisor = 110101, CRC bits?



Transmitted bits

1010001101**01110**

- Data received 101000110101110



All zero means no error, data is accepted

■ Error Correction

Error correction approaches

- **Forward Error Correction (FEC)**: Redundancy bits are transmitted to correct the error in the receiver
  - example: Hamming code
  - add redundancy to transmitted data
  - well suited for channels with high error rate
  - create high overhead

- **Backward Error Correction (BEC):** Sender retransmits the data upon request by the receiver
  - example: Automatic Repeat Request (ARQ)
  - well suited for channels with low error rate
  - request and retransmission cause high delay

- **FEC** and **BEC**
  - BEC is used when FEC cannot correct the error

FEC: Hamming Code

originally designed to detect 2 bit errors and to correct a single bit error now, some hamming code can correct burst errors here, only single bit error correction will be shown

- For m data bits, k parity bits are added
- Total number of bits, n in the code is calculated as $n= 2^k-1$
- Data bits, $m=n- k$
- k parity bits are places in the positions 1, 2, …, 2^k-1

- Transmission end
  - Calculates parity bits as described above
- Receiving end
  - Calculates parity bits in the same way
  - The decimal value of the result indicates the error position

For **m=4** data bits, **k** parity bits are added

Now, how to get the appropriate **k**

Taking different integer of **k**, we need to calculate **n** such that

m=n-k=4

$n=2^k-1=2^1-1=1$ $m=n-k=1-1=0$

$n=2^k-1=2^2-1=3$ $m=n-k=3-2=1$

$n=2^k-1=2^3-1=7$ $m=n-k=7-3=4$

$n=2^k-1=2^4-1=15$ $m=n-k=15-4=11$

- For example, data word, m = 1010

7	6	5	4	3	2	1
D_4	D_3	D_2	P_3	D_1	P_2	P_1
1	0	1		0		

Error Position	Position Number ($C_3C_2C_1$)
0 (no error)	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

- Transmitter sends
  1010010

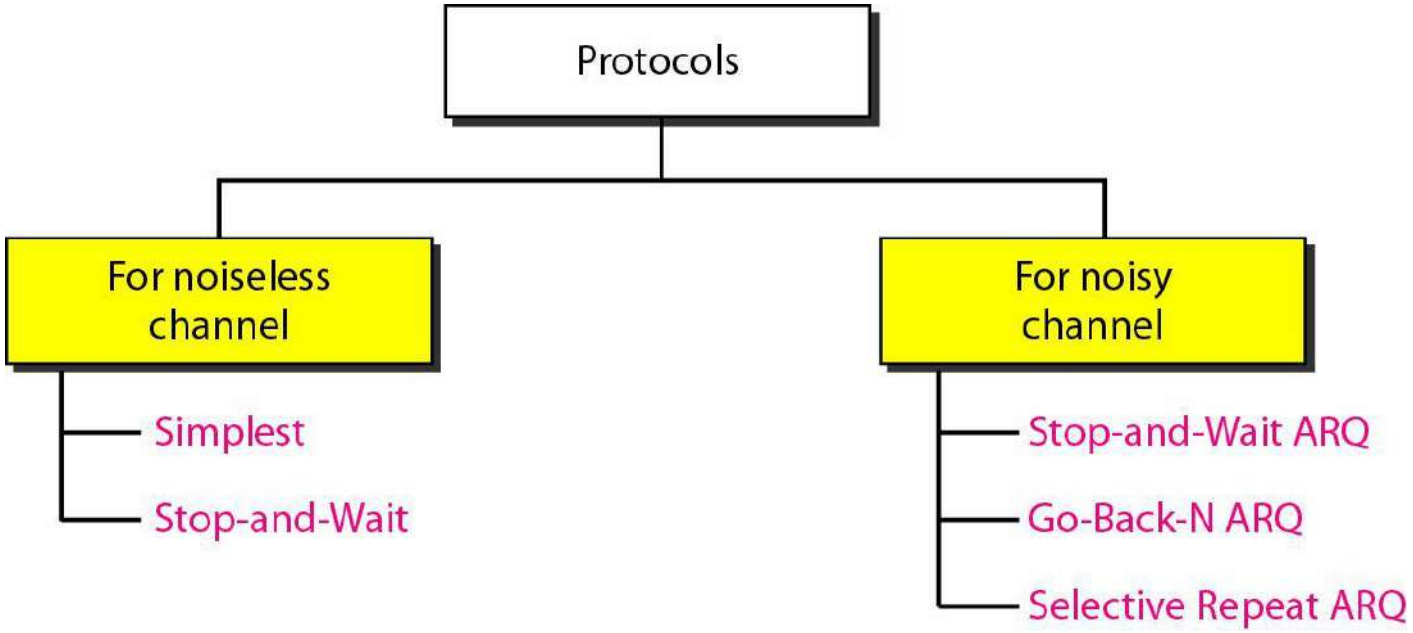
- Transmitter sends
  1010010
- For example, receiver gets
  1110010

- 110 positions number (6) is in error
- Receiver inverts the bit from 1 to 0

■ Q: We need a dataword of at least 7 bits. Calculate values of k and n that satisfy this requirement?

2.3 PROTOCOLS

Now let us see how the data link layer can combine framing, flow control, and error control to achieve the delivery of data from one node to another. The protocols are normally implemented in software by using one of the common programming languages.



■ Noiseless Channels

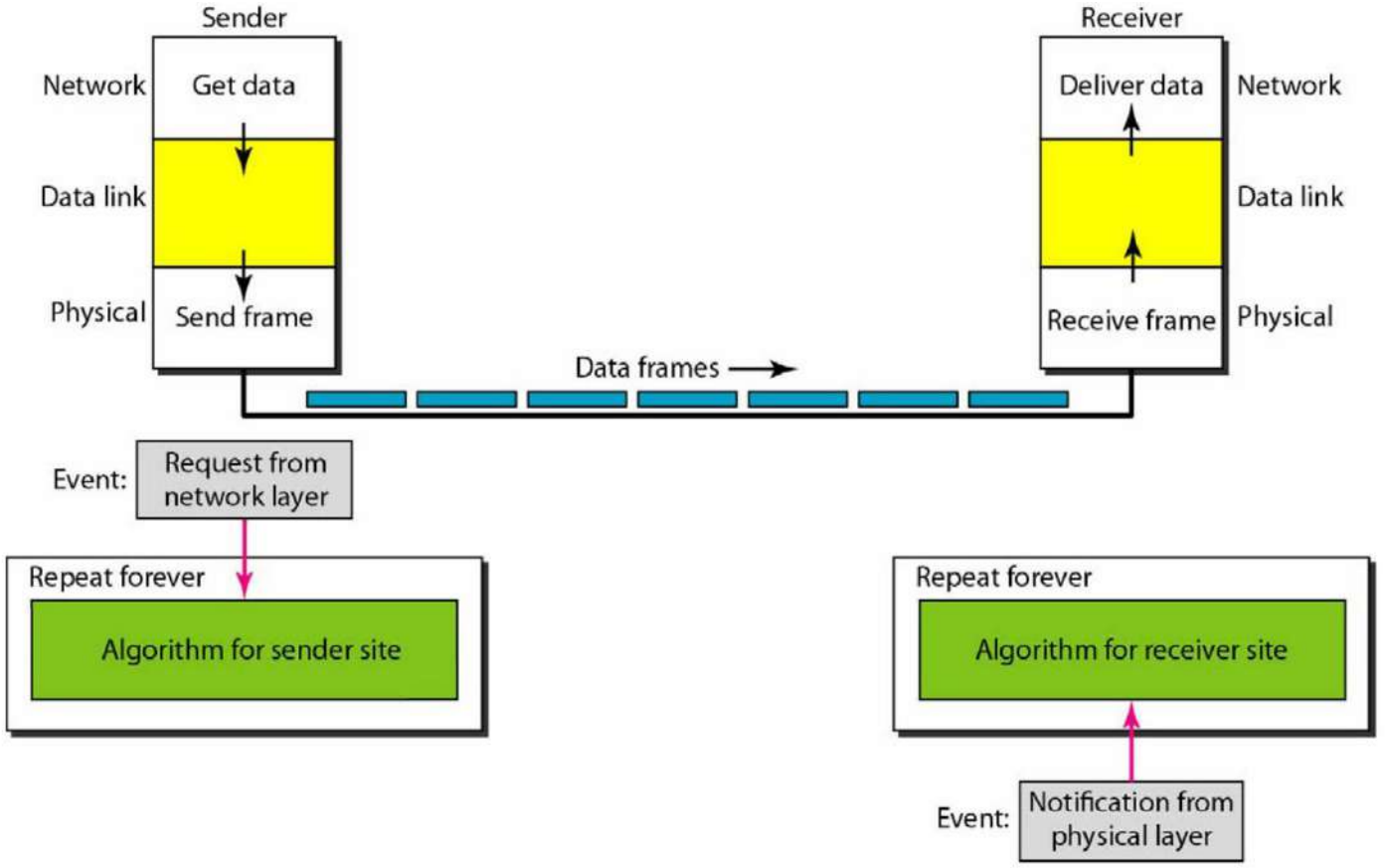
Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel. The first is a protocol that does not use flow control; the second is the one that does. Of course, neither has error control because we have assumed that the channel is a perfect noiseless channel.

■ Simplest Protocol

The Simplest Protocol is one that has no flow or error control. It is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver. We assume that the receiver can immediately handle any frame it receives with a processing time that is small enough to be negligible. The data link layer of the receiver immediately removes the header from the frame and hands the data packet to its network layer, which can also accept the packet immediately. In other words, the receiver can never be overwhelmed with incoming frames.

Design

The data link layer at the sender site gets data from its network layer, makes a frame out of the data, and sends it. The data link layer at the receiver site receives a frame from its physical layer, extracts data from the frame, and delivers the data to its network layer.



Algorithm ***Sender-site algorithm for the simplest protocol***

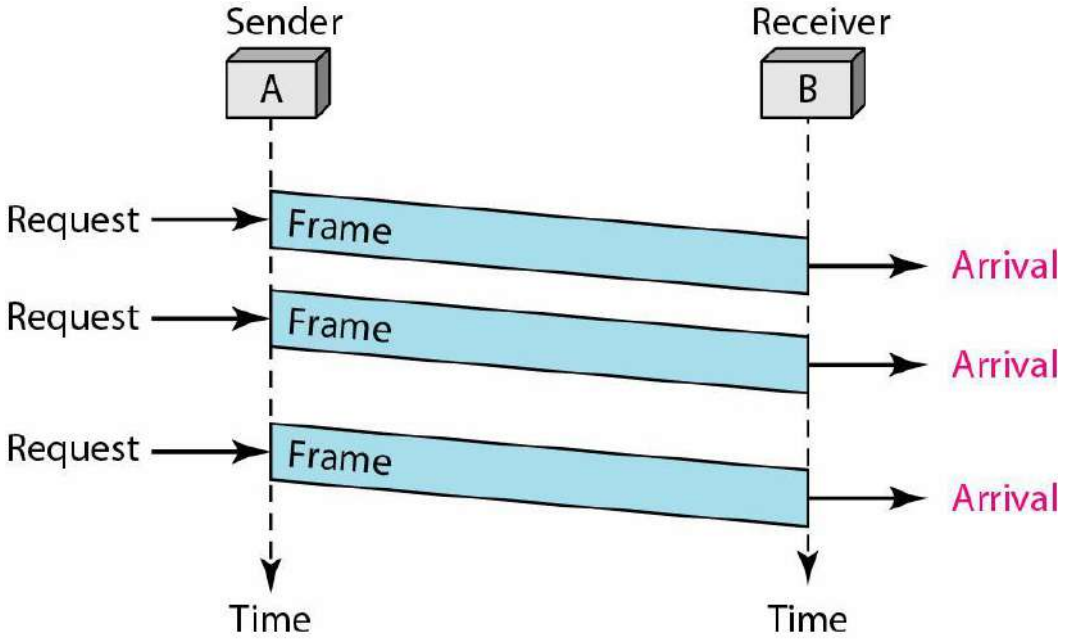
```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(RequestToSend))           //There is a packet to send
5    {
6       GetData();
7       MakeFrame();
8       SendFrame();                    //Send the frame
9    }
10 }
```

Algorithm ***Receiver-site algorithm for the simplest protocol***

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrived
5    {
6       ReceiveFrame();
7       ExtractData();
8       DeliverData();                  //Deliver data to network layer
9    }
10 }
```

Example

The sender sends a sequence of frames without even thinking about the receiver. To send three frames, three events occur at the sender site and three events at the receiver site. Note that the data frames are shown by tilted boxes; the height of the box defines the transmission time difference between the first bit and the last bit in the frame.

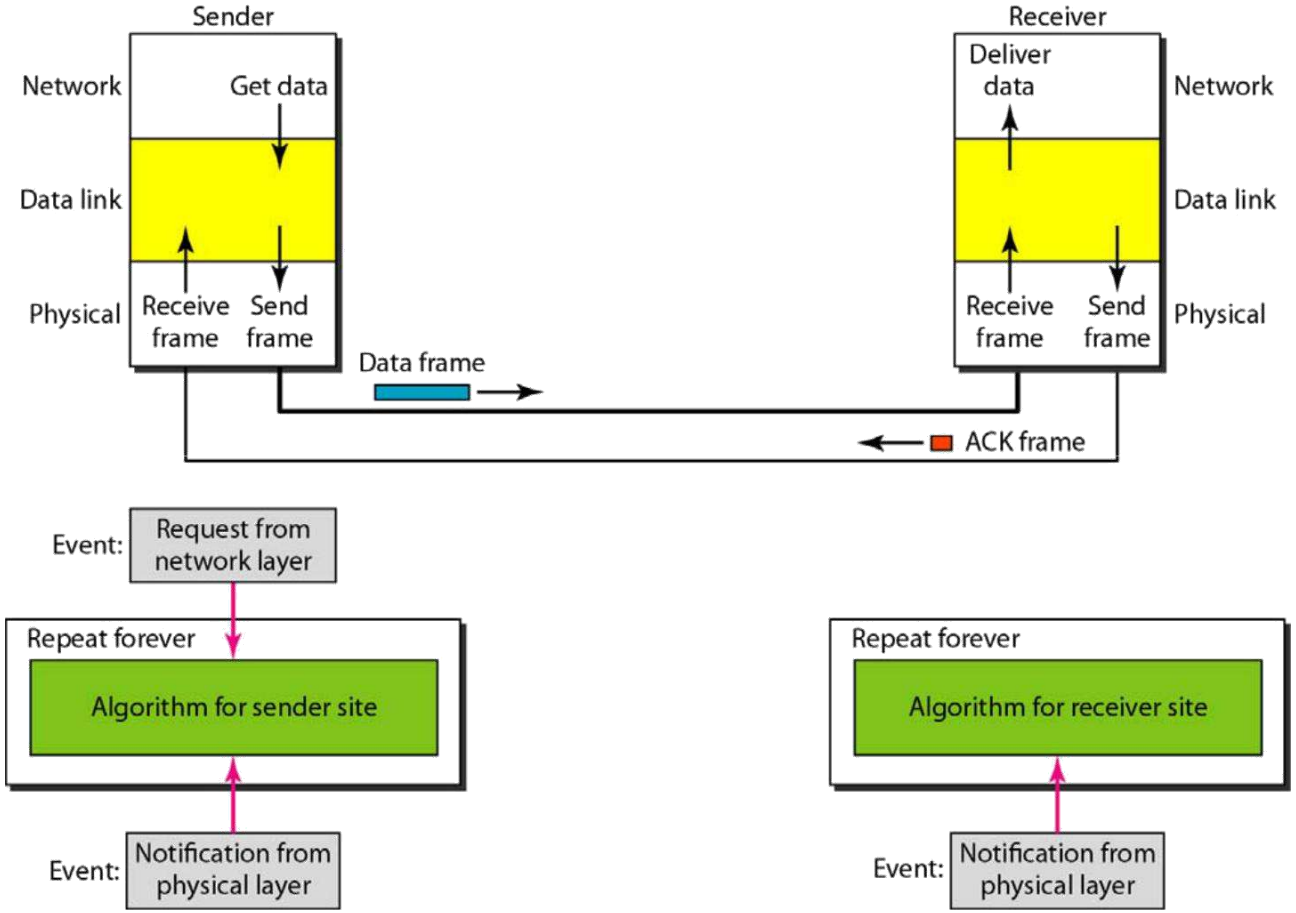


■ Stop-and-Wait Protocol

If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space. There must be feedback from the receiver to the sender. Sender sends one frame, stops until it receives confirmation from the receiver, and then sends the next frame. We add flow control to our previous protocol.

Design

We can see the traffic on the forward channel (from sender to receiver) and the reverse channel. At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel. We therefore need a half-duplex link.


Sender
Receiver
Network
Get data
Deliver data
Network
Data link
Data link
Physical
Receive frame
Send frame
Receive frame
Send frame
Physical
Data frame
ACK frame
Event:
Request from network layer
Repeat forever
Algorithm for sender site
Event:
Notification from physical layer
Repeat forever
Algorithm for receiver site
Event:
Notification from physical layer

Algorithm *Sender-site algorithm for Stop-and-Wait Protocol*

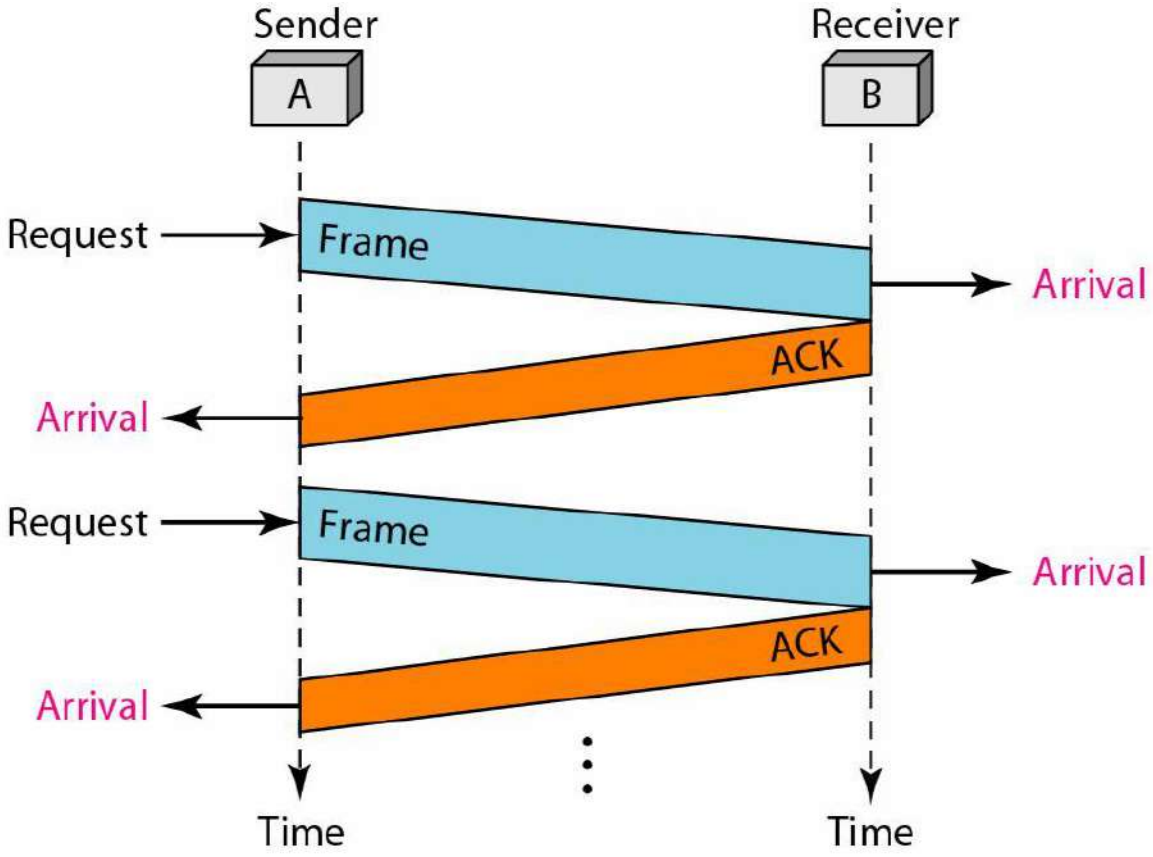
```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

Algorithm *Receiver-site algorithm for Stop-and-Wait Protocol*

```
1  while(true)                          //Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrives
5    {
6       ReceiveFrame();
7       ExtractData();
8       Deliver(data);                  //Deliver data to network layer
9       SendFrame();                    //Send an ACK frame
10   }
11 }
```

Example

The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame. Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.



■ Noisy Channels

Noiseless channels are nonexistent. We can ignore the error, or we need to add error control to our protocols. We discuss three protocols in this section that use error control.

■ Stop-and-Wait Automatic Repeat Request

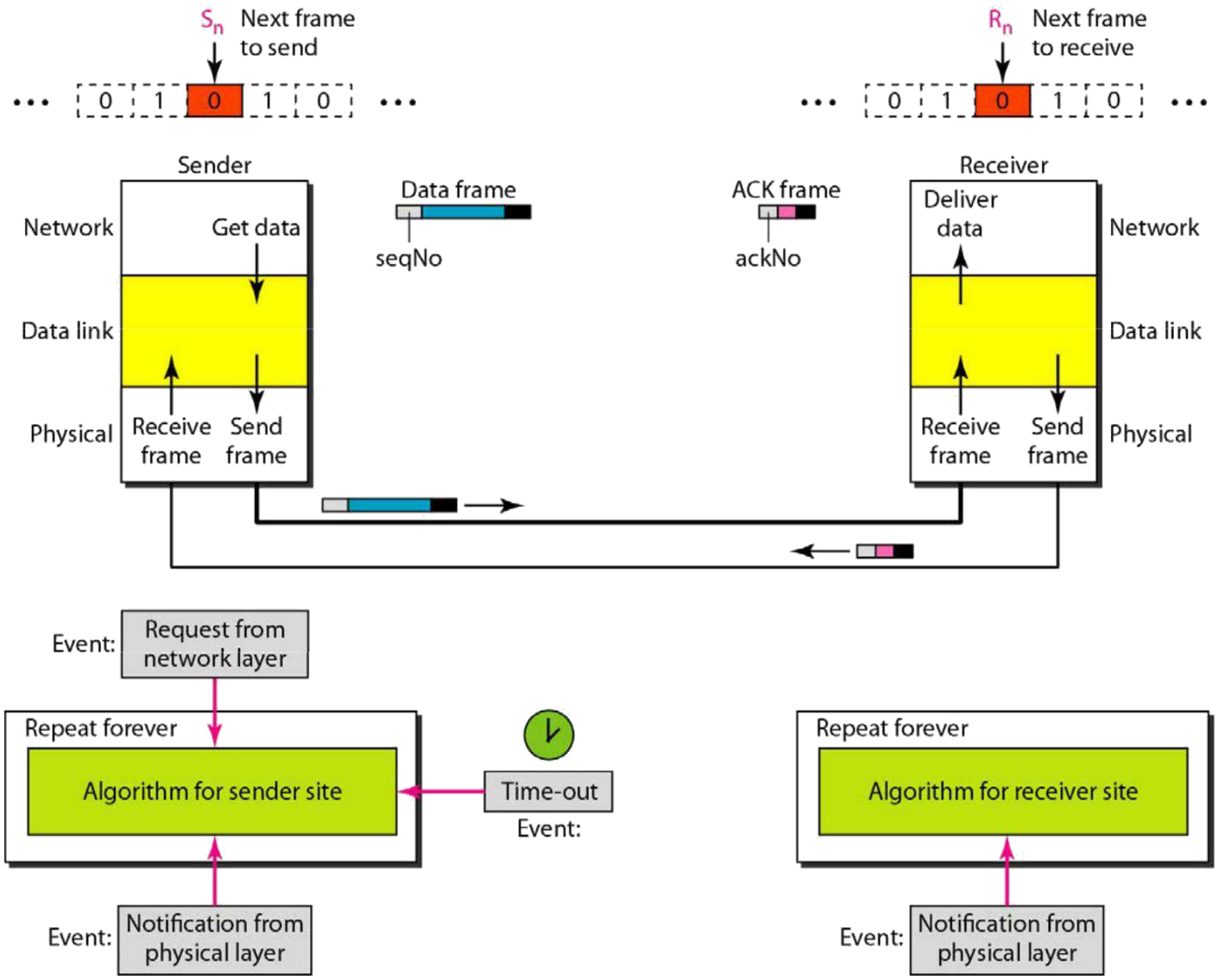
Stop- and-Wait Automatic Repeat Request (Stop-andWait ARQ), adds a simple error control mechanism to the Stop-and-Wait Protocol. To detect and correct corrupted frames, we need to add redundancy bits to our data frame. When the frame arrives at the receiver site, it is checked and if it is corrupted, it is silently discarded.

■ Q: If the receiver does not respond when there is an error, how can the sender know which frame to resend?

- Error correction in Stop-and -Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.
- In Stop-and-Wait ARQ, we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.
- In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Design

The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call S_n (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1). The receiver has a control variable, which we call ***Rn*** (receiver, next frame expected), that holds the number of the next frame expected.


S_n Next frame to send
0 1 0 1 0
R_n Next frame to receive
0 1 0 1 0
Sender
Network
Get data
Data link
Physical
Receive frame
Send frame
Data frame
seqNo
ACK frame
ackNo
Receiver
Deliver data
Network
Data link
Physical
Receive frame
Send frame
Event:
Request from network layer
Repeat forever
Algorithm for sender site
Time-out
Event:
Event:
Notification from physical layer
Repeat forever
Algorithm for receiver site
Event:
Notification from physical layer

Algorithm Sender-site algorithm for Stop-and-Wait ARQ

```
1  Sn = 0;                              // Frame 0 should be sent first
2  canSend = true;                      // Allow the first request to go
3  while(true)                          // Repeat forever
4  {
5    WaitForEvent();                    // Sleep until an event occurs

6    if(Event(RequestToSend) AND canSend)
7    {
8       GetData();
9       MakeFrame(Sn);                  //The seqNo is Sn
10      StoreFrame(Sn);                 //Keep copy
11      SendFrame(Sn);
12      StartTimer();
13      Sn = Sn + 1;
14      canSend = false;
15   }
16   WaitForEvent();                    // Sleep
```

```
17      if(Event(ArrivalNotification)          // An ACK has arrived
18      {
19        ReceiveFrame(ackNo);                 //Receive the ACK frame
20        if(not corrupted AND ackNo == Sn)    //Valid ACK
21          {
22            Stoptimer();
23            PurgeFrame(Sn-1);                //Copy is not needed
24            canSend = true;
25          }
26       }
27
28      if(Event(TimeOut)                      // The timer expired
29      {
30       StartTimer();
31       ResendFrame(Sn-1);                    //Resend a copy check
32      }
33  }
```

Algorithm Receiver-site algorithm for Stop-and-Wait ARQ Protocol

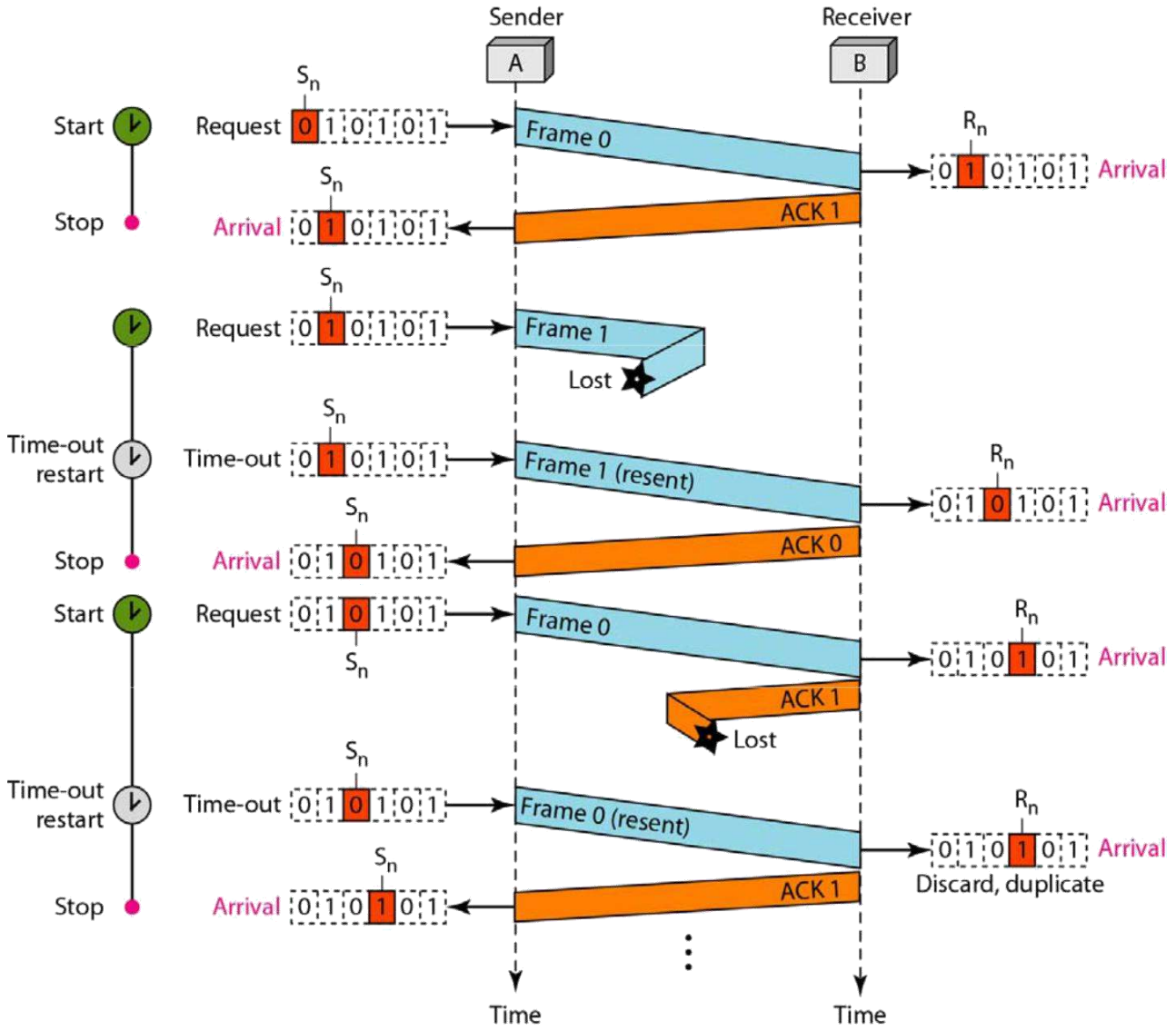
```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7       ReceiveFrame();
8       if(corrupted(frame));
9          sleep();
10      if(seqNo == Rn)              //Valid data frame
11      {
12       ExtractData();
13        DeliverData();             //Deliver data
14        Rn = Rn + 1;
15      }
16       SendFrame(Rn);              //Send an ACK
17   }
18 }
```

Efficiency

The Stop-and-WaitARQ is very inefficient if our channel is thick and long. By thick, we mean that our channel has a large bandwidth; by long, we mean the round-trip delay is long. The product of these two is called the bandwidthdelay product. We can think of the channel as a pipe. The bandwidth-delay product then is the volume of the pipe in bits. The pipe is always there. If we do not use it, we are inefficient. The bandwidth-delay product is a measure of the number of bits we can send out of our system while waiting for news from the receiver.

Example

Frame 0 is sent and acknowledged. Frame 1 is lost and resent after the time-out. The resent frame 1 is acknowledged and the timer stops. Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.


Sender
Receiver
A
B
Start
Stop
Request
Arrival
Frame 0
ACK 1
Frame 1
Lost
Time-out
restart
Time-out
Frame 1 (resent)
ACK 0
Frame 0
ACK 1
Lost
Frame 0 (resent)
Discard, duplicate
Time
Time

Example

Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again. However, the system sends only 1000 bits. We can say that the link utilization is only 1000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.

Pipelining

In networking and in other areas, a task is often begun before the previous task has ended. This is known as pipelining. There is no pipelining in Stop-and-Wait ARQ because we need to wait for a frame to reach the destination and be acknowledged before the next frame can be sent.

■ Go-Back-N Automatic Repeat Request

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment. In other words, we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In this section, we discuss one protocol that can achieve this goal.
In this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive.

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 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence numbers are

0, 1,2,3,4,5,6, 7,8,9, 10, 11, 12, 13, 14, 15,0, 1,2,3,4,5,6,7,8,9,10, 11, ...

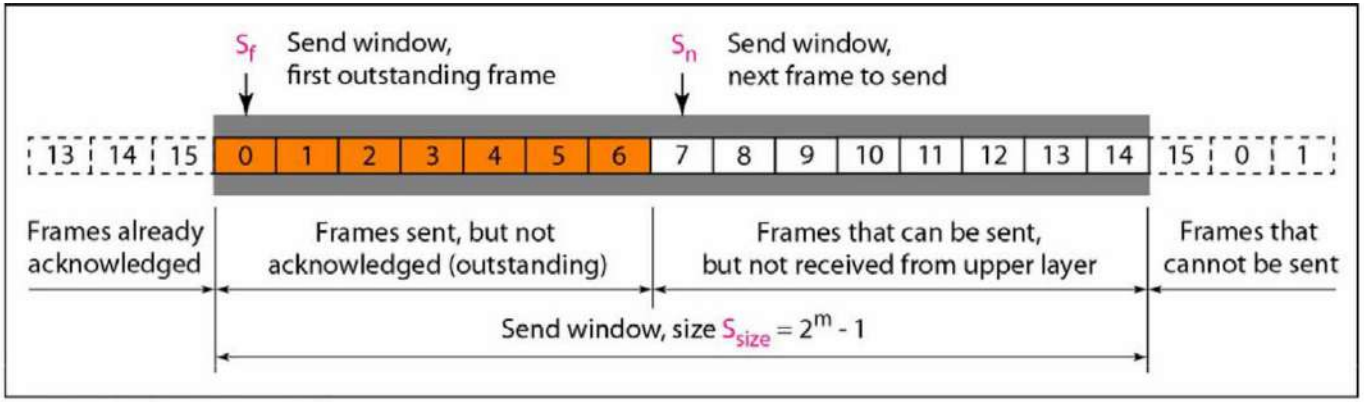
In other words, the sequence numbers are modulo-2^m.

- **In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.**

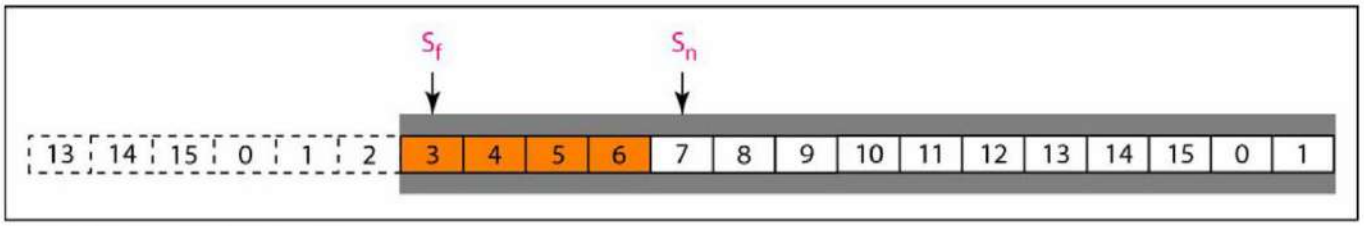
Sliding Window

In this protocol (and the next), the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called receive sliding window.

Send window for Go-Back-N ARQ



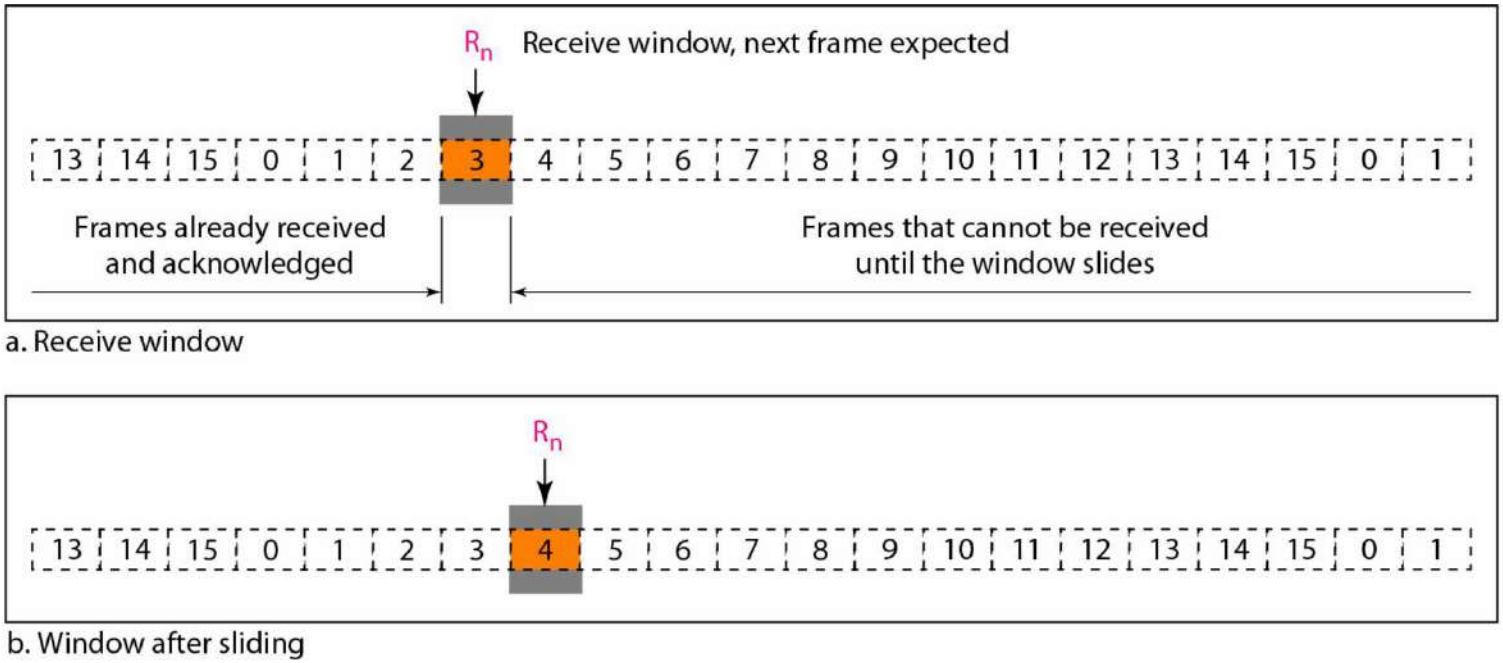
a. Send window before sliding



b. Send window after sliding

- The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}.
- The send window can slide one or more slots when a valid acknowledgment arrives.

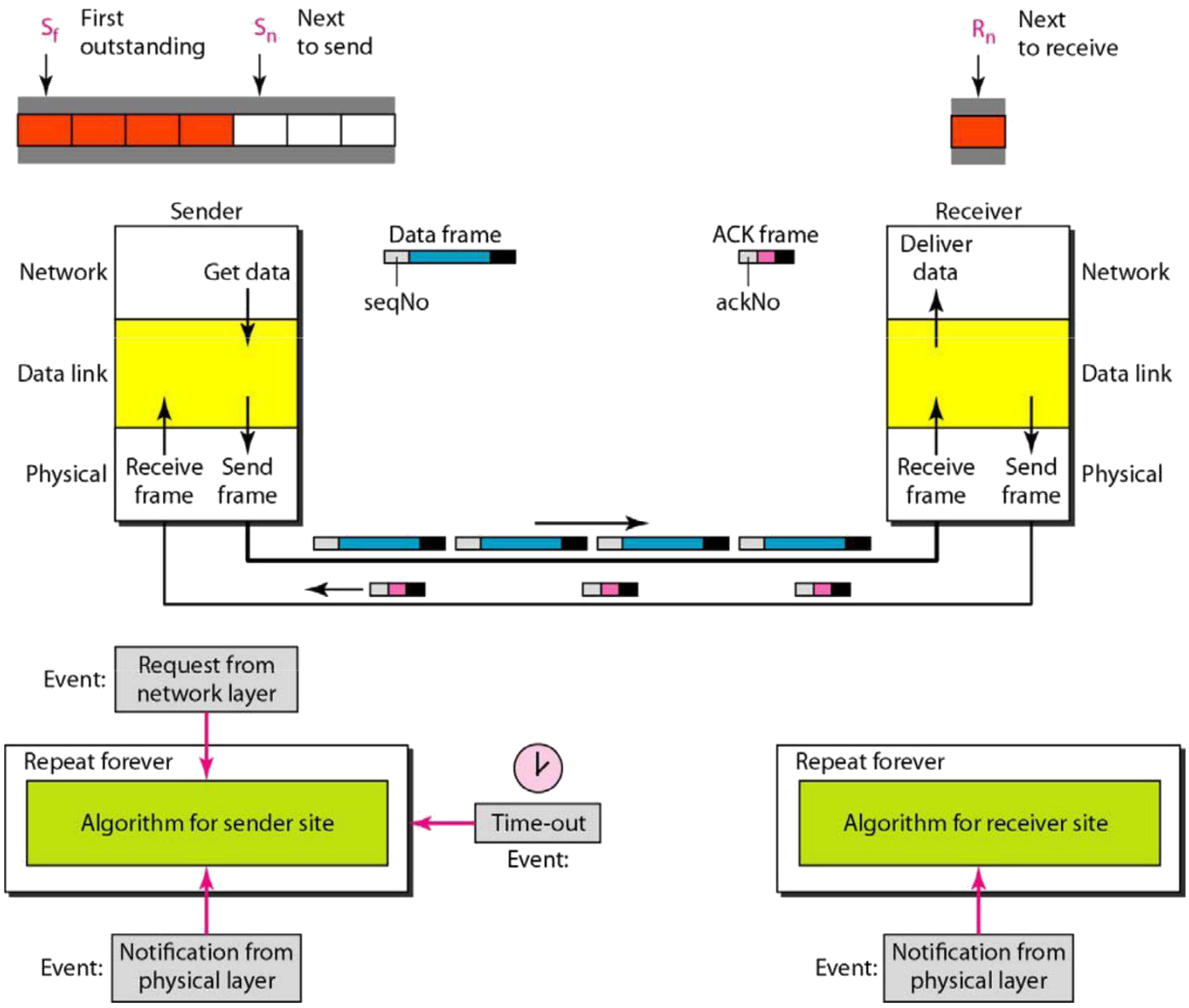
Receive window for Go-Back-N ARQ



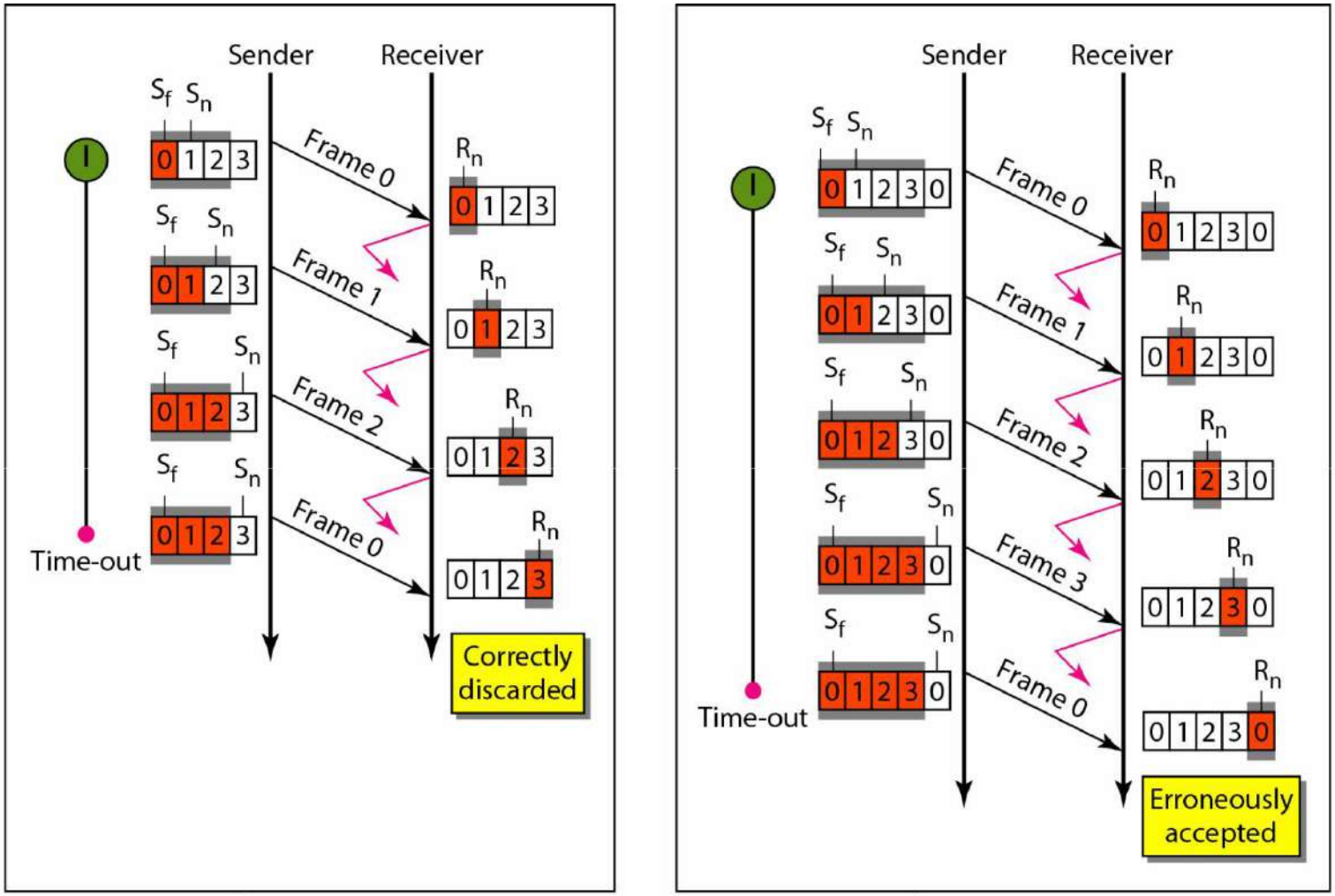
- **The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n. The window slides when a correct frame has arrived; sliding occurs one slot at a time.**

Design

Multiple frames can be in transit in the forward direction, and multiple acknowledgments in the reverse direction. The idea is similar to Stop-and-Wait ARQ; the difference is that the send window allows us to have as many frames in transition as there are slots in the send window.


S_f First outstanding
S_n Next to send
R_n Next to receive
Sender
Receiver
Network
Get data
Data link
Physical
Receive frame
Send frame
Data frame
seqNo
ACK frame
ackNo
Deliver data
Network
Data link
Physical
Receive frame
Send frame
Event:
Request from network layer
Repeat forever
Algorithm for sender site
Time-out
Event:
Event:
Notification from physical layer
Repeat forever
Algorithm for receiver site
Event:
Notification from physical layer

■ **In Go- Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.**



a. Window size < 2^m

b. Window size = 2^m

Algorithm Go-Back-N sender algorithm

```
1  Sw = 2^m - 1;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7   WaitForEvent();
8    if(Event(RequestToSend))            //A packet to send
9    {
10      if(Sn-Sf >= Sw)                  //If window is full
11            Sleep();
12      GetData();
13      MakeFrame(Sn);
14      StoreFrame(Sn);
15      SendFrame(Sn);
16      Sn = Sn + 1;
17      if(timer not running)
18           StartTimer();
19   }
20
```

```
21    if(Event(ArrivalNotification))  //ACK arrives
22    {
23       Receive(ACK);
24       if(corrupted(ACK))
25            Sleep();
26       if((ackNo>Sf)&&(ackNo<=Sn))  //If a valid ACK
27       While(Sf <= ackNo)
28        {
29         PurgeFrame(Sf);
30         Sf = Sf + 1;
31        }
32        StopTimer();
33    }
34
35    if(Event(TimeOut))               //The timer expires
36    {
37     StartTimer();
38     Temp = Sf;
39     while(Temp < Sn);
40       {
41        SendFrame(Sf);
42        Sf = Sf + 1;
43       }
44    }
45 }
```

Algorithm Go-*Back-N receiver algorithm*

```
1  Rn = 0;
2
3  while (true)                          //Repeat forever
4  {
5    WaitForEvent();
6
7    if(Event(ArrivalNotification)) /Data frame arrives
8    {
9       Receive(Frame);
10      if(corrupted(Frame))
11           Sleep();
12      if(seqNo == Rn)                  //If expected frame
13      {
14        DeliverData();                 //Deliver data
15        Rn = Rn + 1;                   //Slide window
16        SendACK(Rn);
17      }
18   }
19 }
```

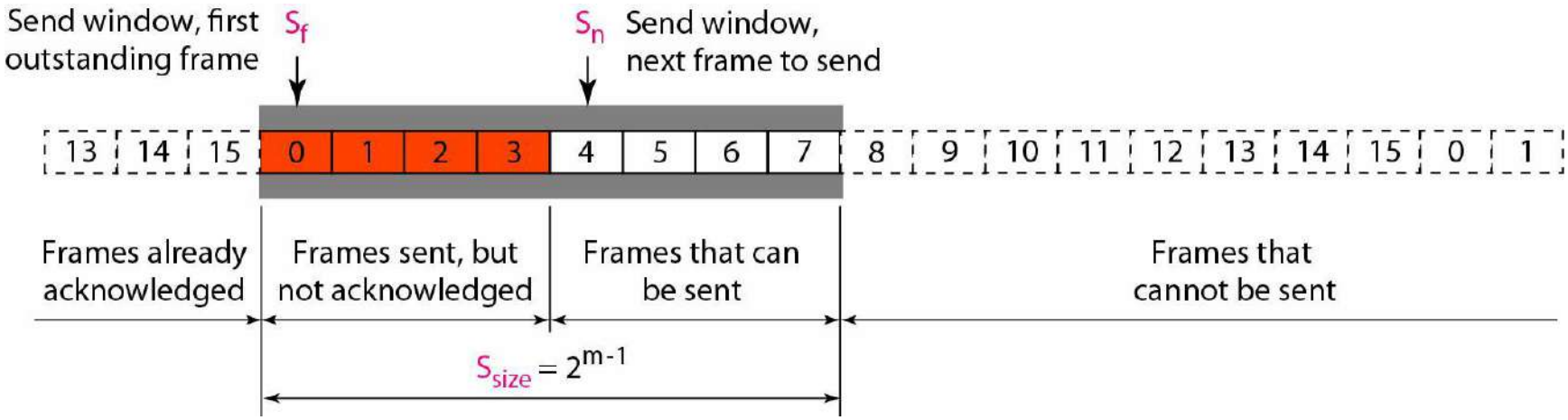
■ Selective Repeat Automatic Repeat Request

In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission. For noisy links, there is another mechanism that does not resend ***N*** frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called **Selective RepeatARQ**. It is more efficient for noisy links, but the processing at the receiver is more complex.

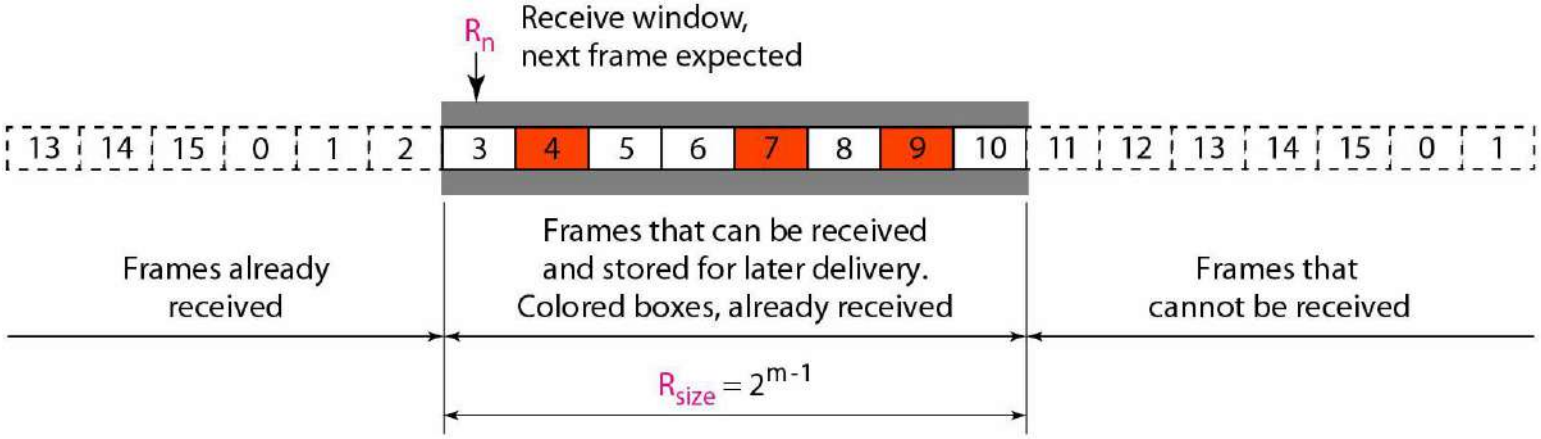
Windows

The Selective Repeat Protocol also uses two windows: a send window and a receive window. However, there are differences between the windows in this protocol and the ones in Go -Back-N. First, the size of the send window is much smaller; it is 2^m- 1. Second, the receive window is the same size as the send window.

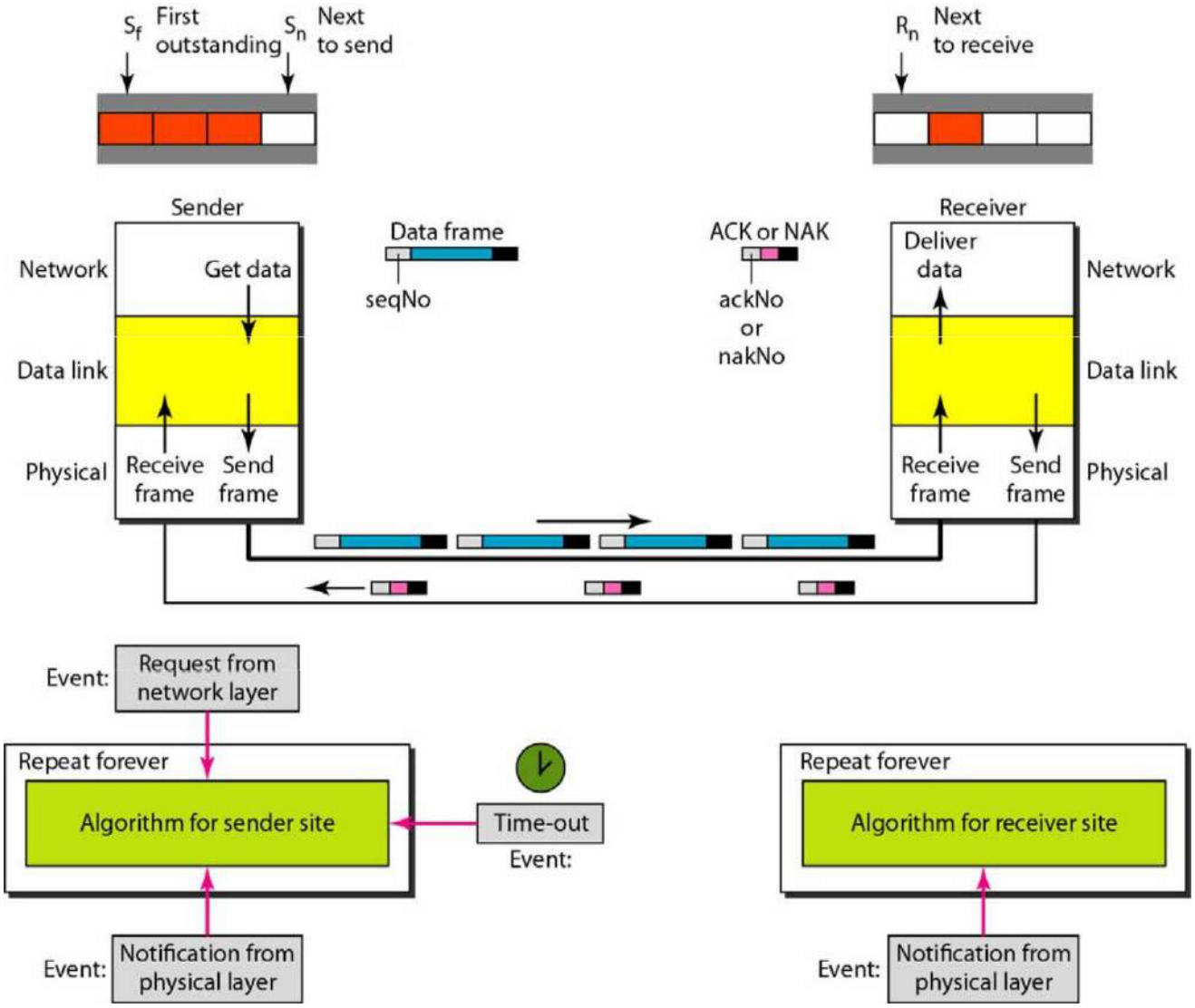
Send window for Selective Repeat ARQ



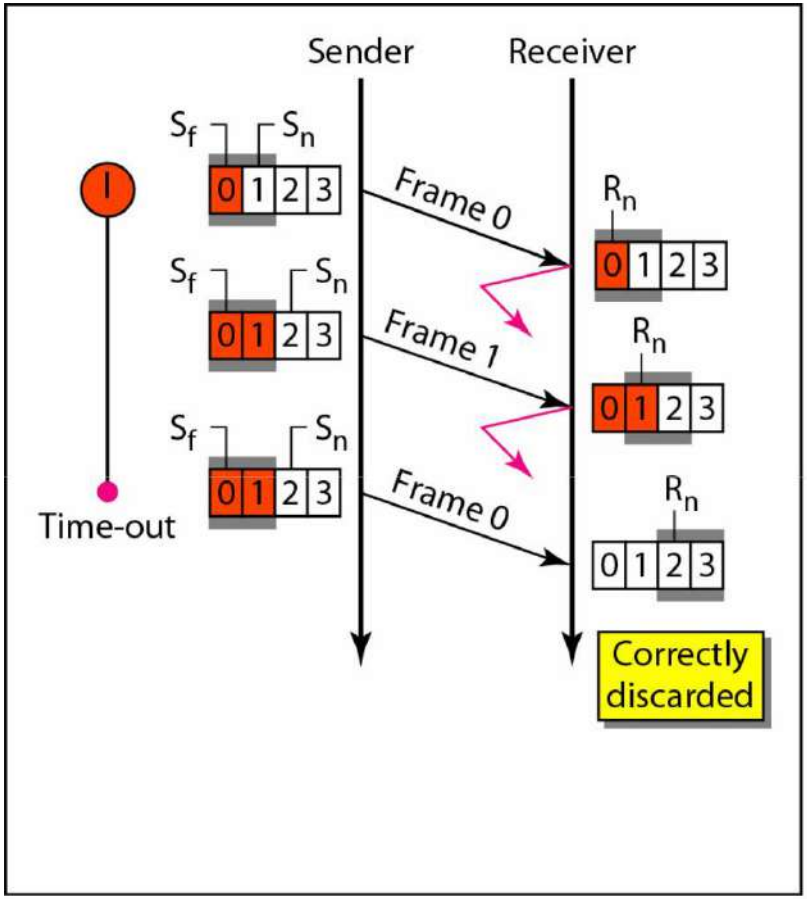
Receive window for Selective Repeat ARQ



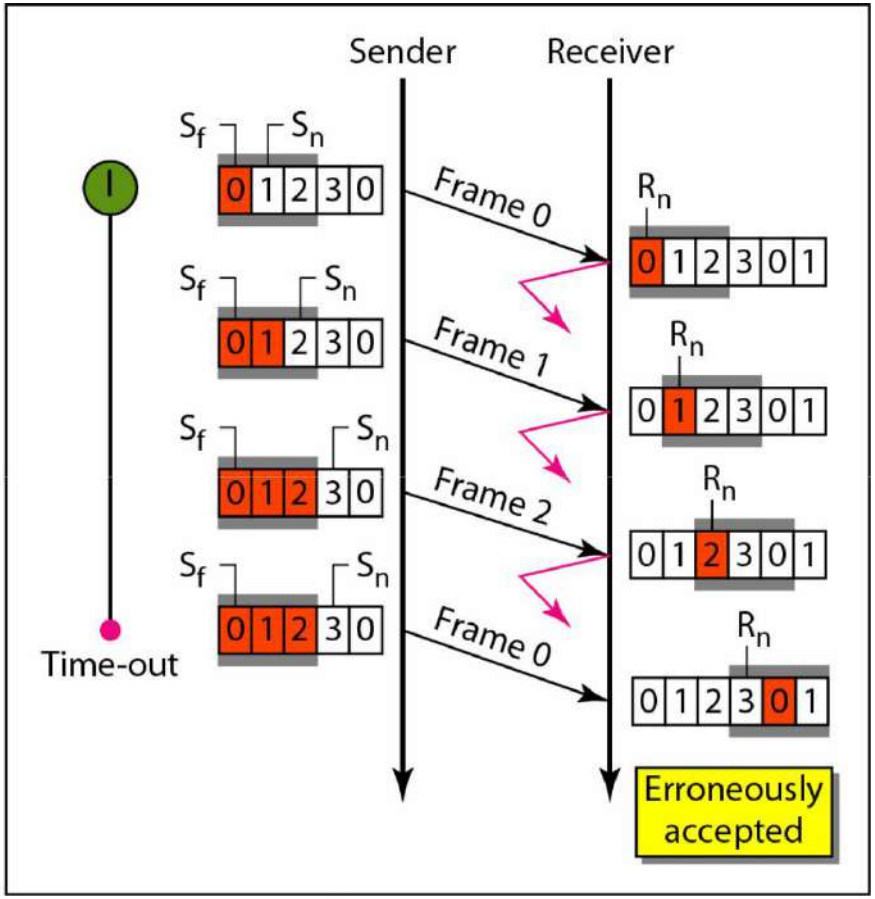
Design of Selective Repeat ARQ



Selective Repeat ARQ, window size



a. Window size = 2^{m-1}



b. Window size > 2^{m-1}

■ **In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.**

Algorithm ***Sender-site Selective Repeat algorithm***

```
1  Sw = 2^(m-1) ;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7    WaitForEvent();
8    if(Event(RequestToSend))            //There is a packet to send
9    {
10     if(Sn-Sf >= Sw)                   //If window is full
11           Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     StartTimer(Sn);
18   }
19
```

```
20  if(Event(ArrivalNotification)) //ACK arrives
21  {
22     Receive(frame);              //Receive ACK or NAK
23     if(corrupted(frame))
24          Sleep();
25     if (FrameType == NAK)
26        if (nakNo between Sf and Sn)
27        {
28         resend(nakNo);
29         StartTimer(nakNo);
30        }
31     if (FrameType == ACK)
32        if (ackNo between Sf and Sn)
33        {
34          while(sf < ackNo)
35          {
36           Purge(sf);
37           StopTimer(sf);
38           Sf = Sf + 1;
39          }
40        }
41  }
```

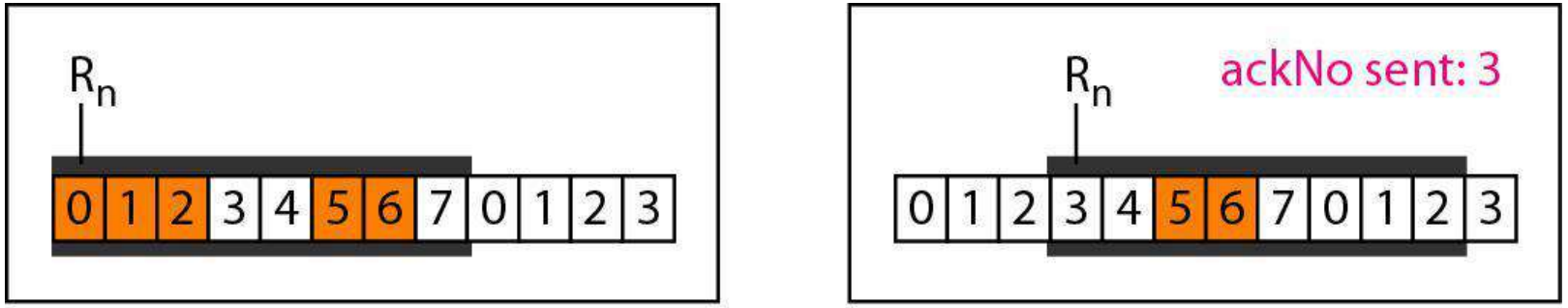
```
42
43    if(Event(TimeOut(t)))              //The timer expires
44    {
45     StartTimer(t);
46     SendFrame(t);
47    }
48  }
```

Algorithm Receiver-site Selective Repeat algorithm

```
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5      Marked(slot) = false;
6
7  while (true)                                  //Repeat forever
8  {
9    WaitForEvent();
10
11   if(Event(ArrivalNotification))              /Data frame arrives
12   {
13      Receive(Frame);
14      if(corrupted(Frame))&& (NOT NakSent)
15      {
16       SendNAK(Rn);
17       NakSent = true;
18       Sleep();
19      }
20      if(seqNo <> Rn)&& (NOT NakSent)
21      {
22       SendNAK(Rn);
```

```
23          NakSent = true;
24          if ((seqNo in window)&&(!Marked(seqNo))
25          {
26           StoreFrame(seqNo)
27           Marked(seqNo)= true;
28           while(Marked(Rn))
29           {
30            DeliverData(Rn);
31            Purge(Rn);
32            Rn = Rn + 1;
33            AckNeeded = true;
34           }
35            if(AckNeeded);
36            {
37            SendAck(Rn);
38            AckNeeded = false;
39            NakSent = false;
40            }
41          }
42         }
43    }
44  }
```

Delivery of data in Selective Repeat ARQ



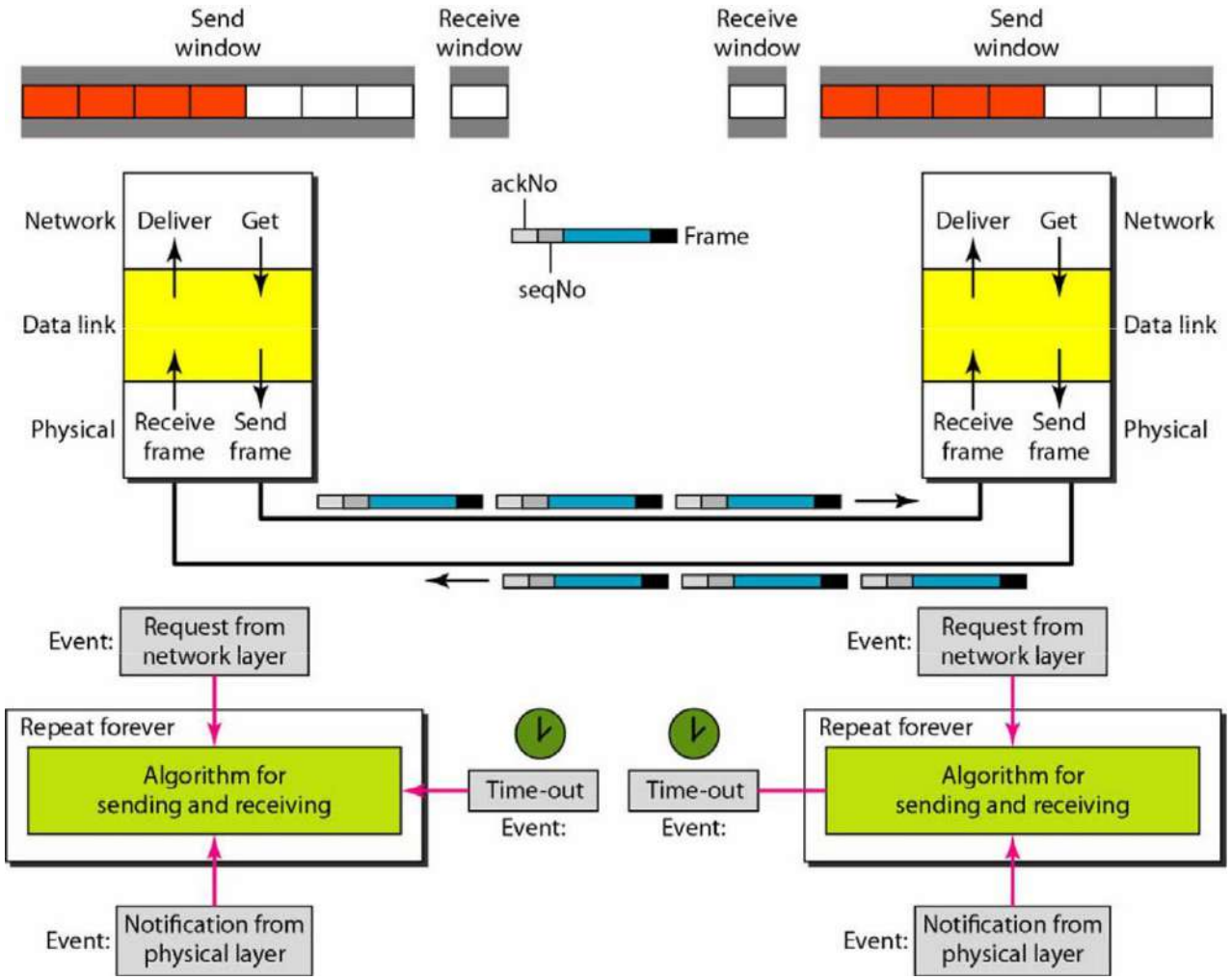
a. Before delivery

b. After delivery

- **Piggybacking**

In real life, data frames are normally flowing in both directions: from node A to node B and from node B to node A. This means that the control information also needs to flow in both directions. A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols.

Design of piggybacking in Go-Back-N ARQ



■ HDLC

High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the ARQ mechanisms.

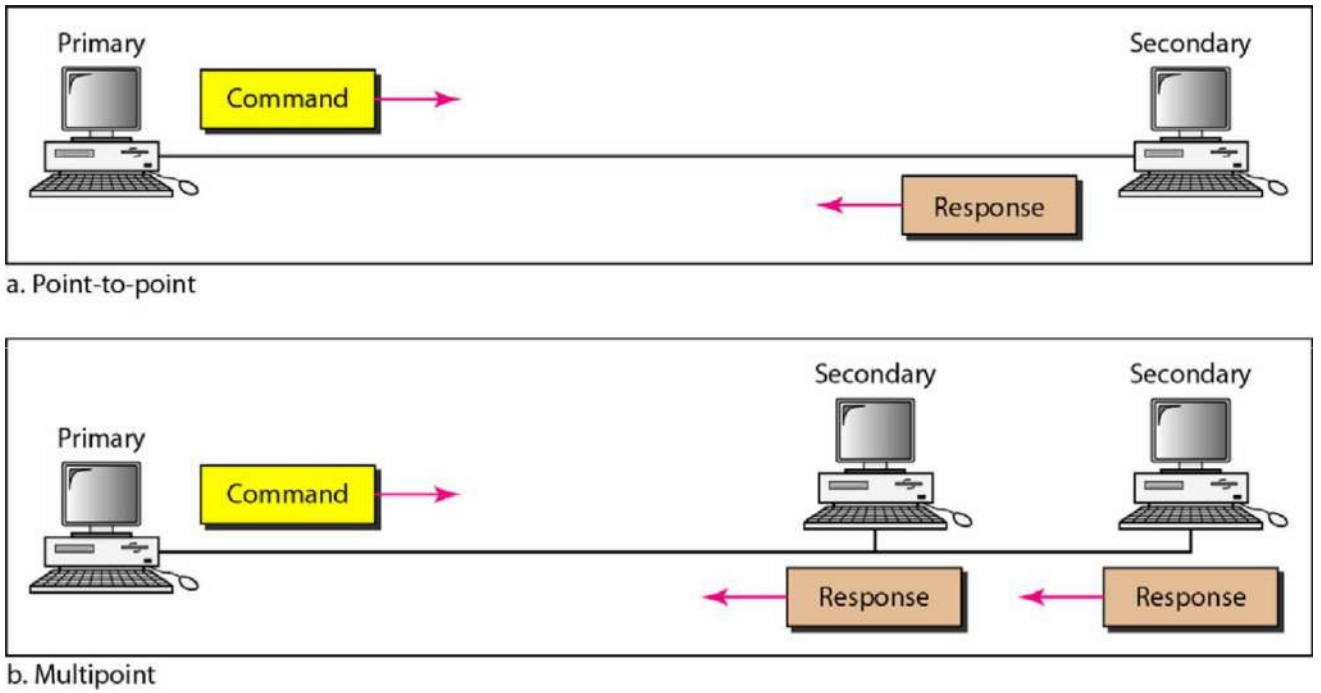
Configurations and Transfer Modes

HDLC provides two common transfer modes that can be used in different configurations: normal response mode (NRM) and asynchronous balanced mode (ABM).

Normal Response Mode

In normal response mode (NRM), the station configuration is unbalanced. We have one primary station and multiple secondary stations. A primary station can send commands; a secondary station can only respond.

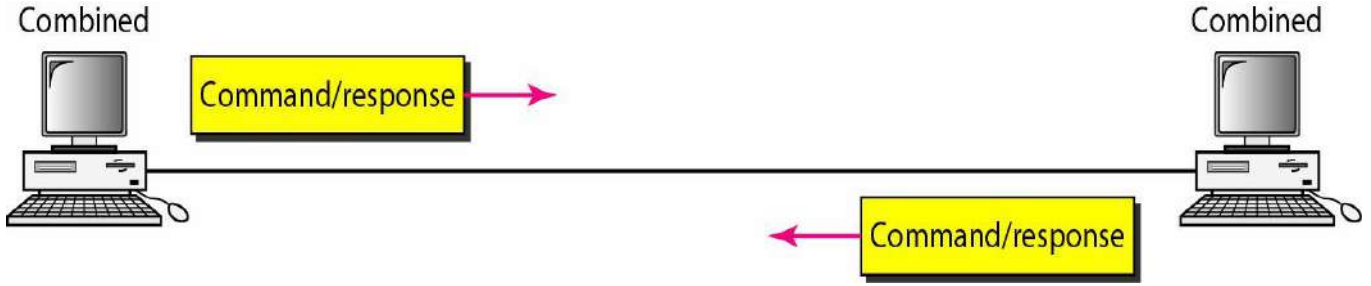
Normal response mode



Asynchronous Balanced Mode

In asynchronous balanced mode (ABM), the configuration is balanced. The link is point-to -point, and each station can function as a primary and a secondary (acting as peers). This is the common mode today.

Asynchronous balanced mode



Frames

To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames: information frames (I-frames), supervisory frames (S-frames), and unnumbered frames (V-frames). Each type of frame serves as an envelope for the transmission of a different type of message.

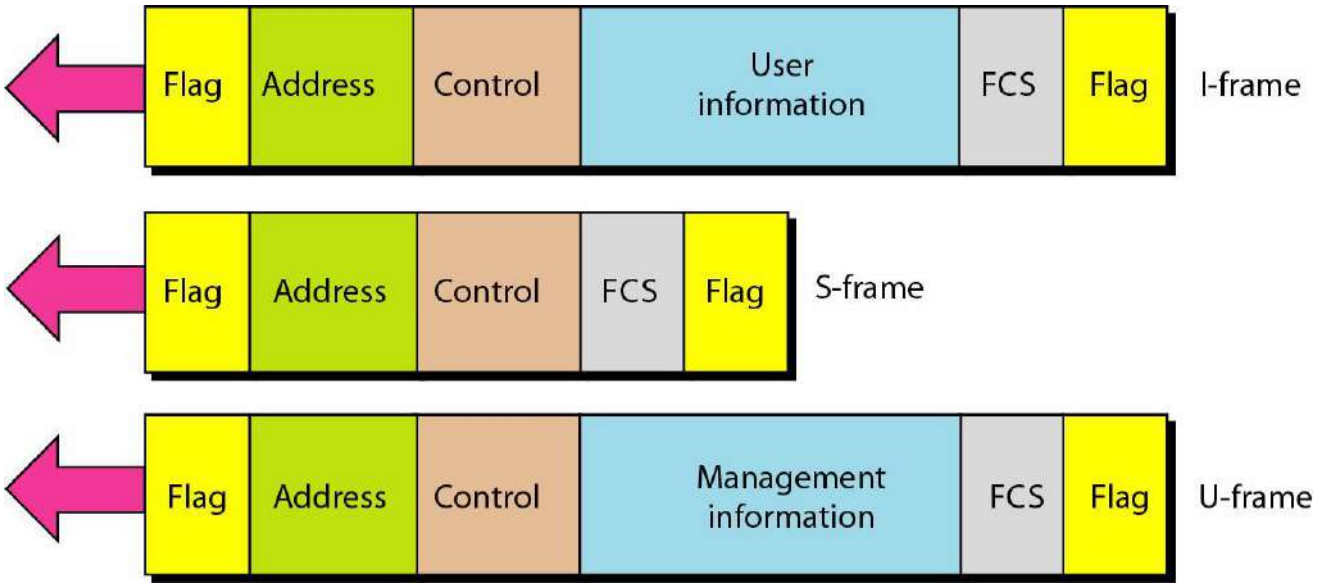
I-frames are used to transport user data and control information relating to user data (piggybacking).

S-frames are used only to transport control information.

V-frames are reserved for system management. Information carried by V-frames is intended for managing the link itself.

Frame Format

Each frame in HDLC may contain up to six fields



Control Field

The control field determines the type of frame and defines its functionality. The format is specific for the type of frame,