데이터통신시스템

서강대학교
소재우

4. Data Link Control(1/2)

학습에 앞서

■ 학습개요

– **데이터링크 제어의 필요성과** ARQ **프로토콜의 동작을 학습한다**.

■ 학습목표

– ARQ **프로토콜 동작을 학습하고 장단점을 이해한다**.
– Stop-and-Wait ARQ, Go-Back-N ARQ, Selective Repeat ARQ **프로토콜 동작을 학습한다**.

1. Framing

The data link layer needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter.

Topics discussed in this section:

Fixed-Size Framing
Variable-Size Framing

1. Framing

- 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.
- Frames can be of fixed or variable size.

- Fixed-Size Framing
  - There is no need of defining the boundaries of the frames.
  - This size itself can be used as a delimiter.
  - Example: ATM wide-are network, which uses frames of fixed size called cells (5+48=53 bytes).

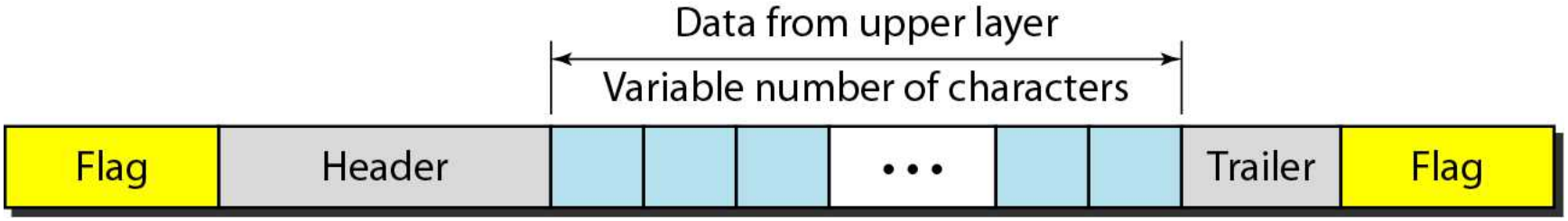
- Variable-Size Framing
  - Need a way to define the end of the frame and the beginning of the next.
  - Example: prevalent in local-area network
  - Two approaches: character-oriented protocols, Bit-oriented protocols

1. Framing

Character-Oriented Protocols

- Data to be carried are 8-bit characters from a coding system such as ASCII.
  - Header: normally carries the source and destination address and other control information.
  - Trailer: carries error detection or error correction redundant bits.
  - Flag: To separate one frame from the next, an 8-bit (1 byte) flag is added at the beginning and the end of a frame.

Figure 11.1 *A frame in a character-oriented protocol*



1. Framing

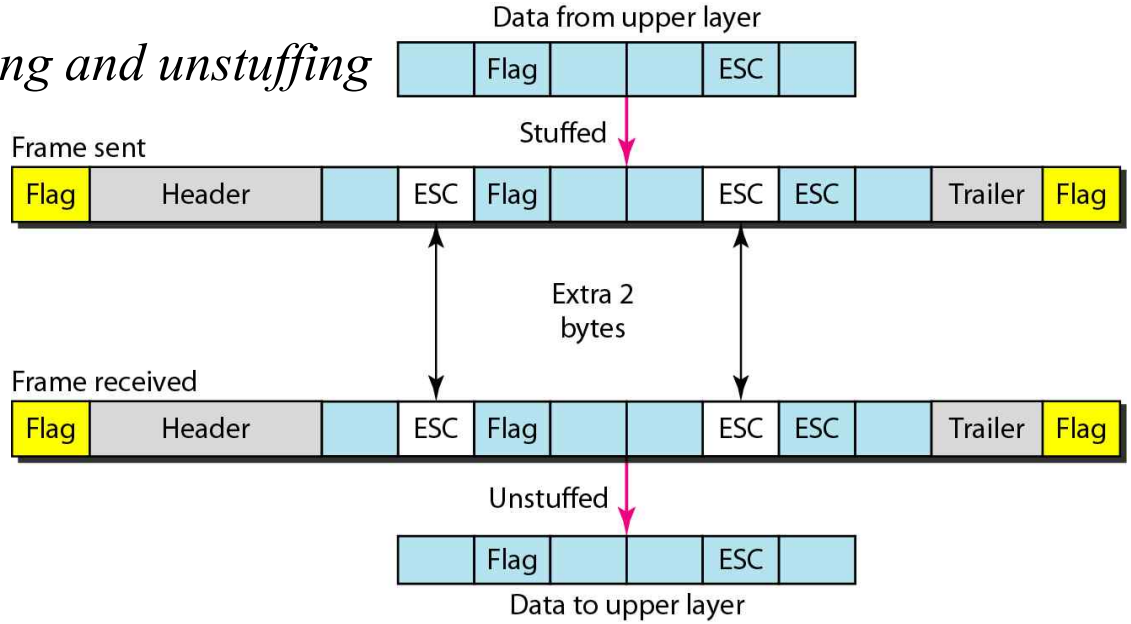
- Byte-stuffing

– The flag could be selected to be any character not used for text communication. Any pattern used for the flag could also be part of the information. → The receiver thinks it has reached the end of the frame.

– To fix this problem, a byte-stuffing strategy was added to character-oriented framing. A special byte, usually called the escape character (ESC), is added to the data section of the frame when there is a character with the same pattern as the flag.

Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.

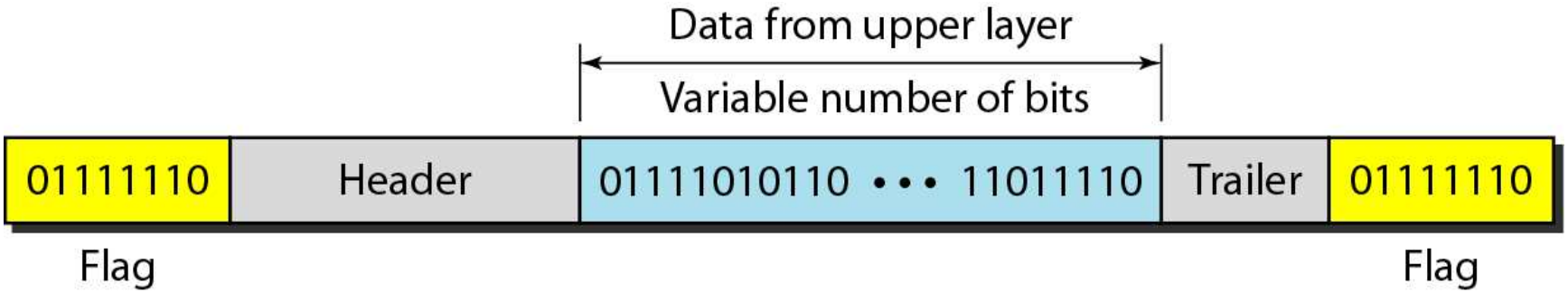
Figure 11.2 *Byte stuffing and unstuffing*



Bit-Oriented Protocols

- 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.
- Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame.

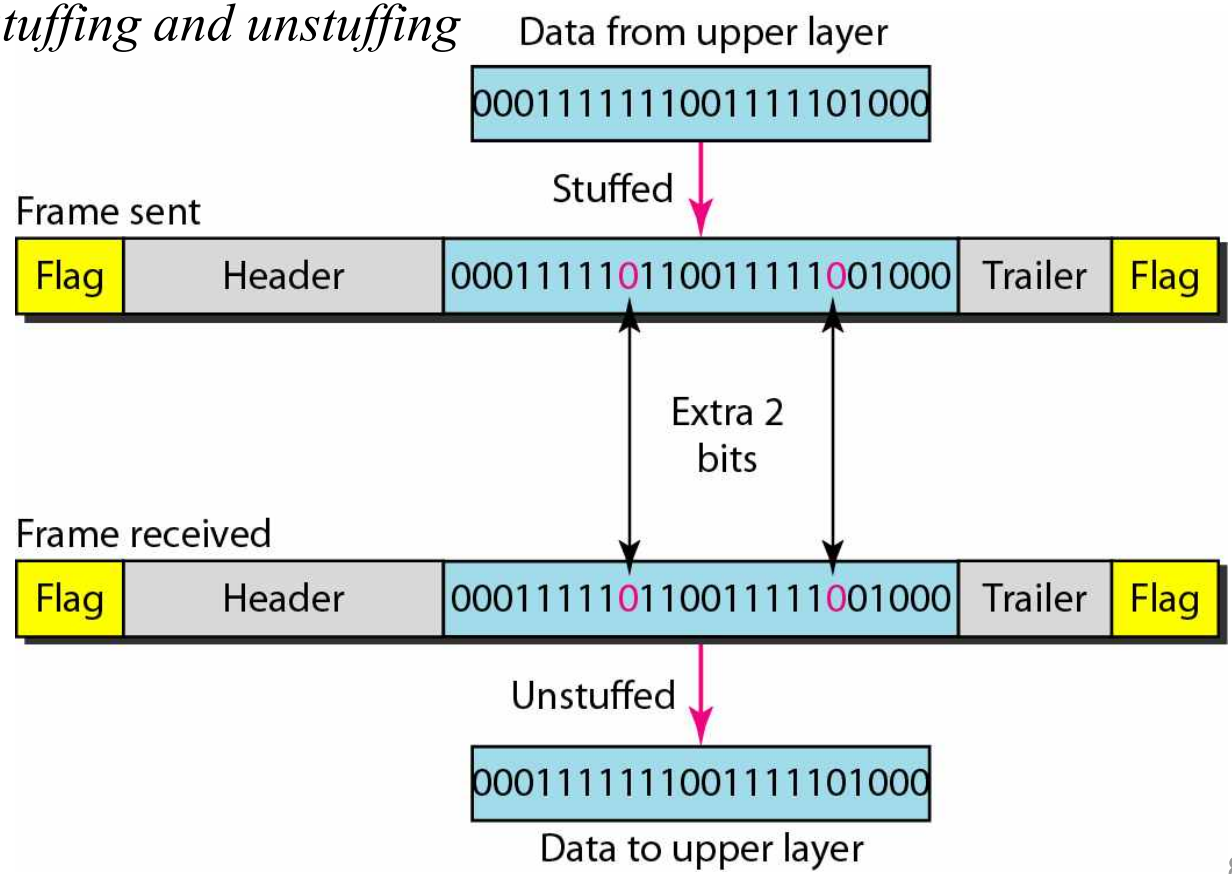
Figure 11.3 *A frame in a bit-oriented protocol*



- Bit stuffing

Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.

Figure 11.4 *Bit stuffing and unstuffing*



2. Flow And Error Control

The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.

Topics discussed in this section:

Flow Control
Error Control

2. Flow And Error Control

- Flow Control
  - Flow control coordinates the amount of data that can be sent before receiving an acknowledgement.
  - The flow of data must not be allowed to overwhelm the receiver.
  - Each receiving device has a block of memory, called a buffer, reserved for storing incoming data until they proceed. 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.
  - Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (**ARQ**).

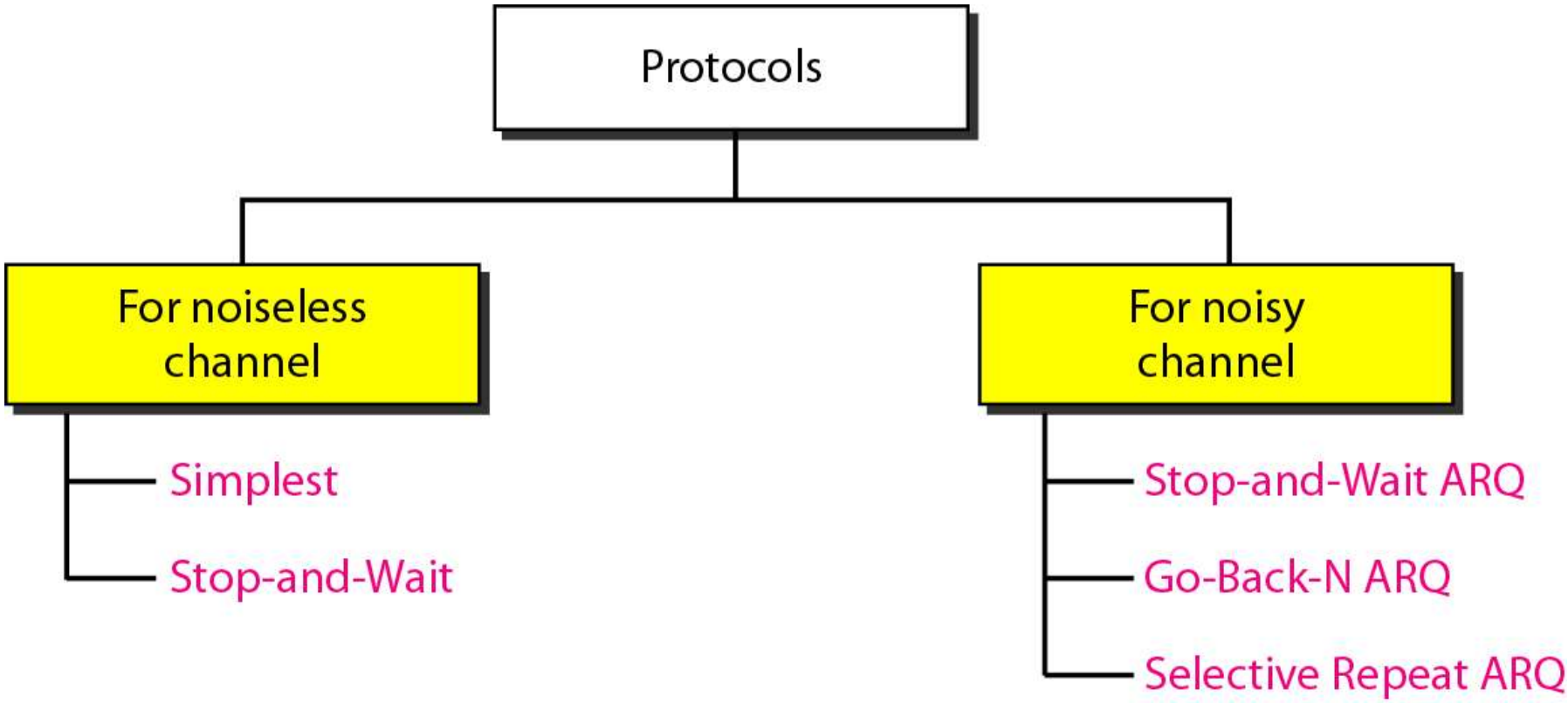
Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.

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. To make our discussions language-free, we have written in pseudocode a version of each protocol that concentrates mostly on the procedure instead of delving into the details of language rules.

Figure 11.5 *Taxonomy of protocols discussed in this chapter*



4. 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 does not use flow control*
- *The second is the one that does.*

Because of a perfect noiseless channel, neither has error control.

Topics discussed in this section:

Simplest Protocol
Stop-and-Wait Protocol

4. Noiseless Channels

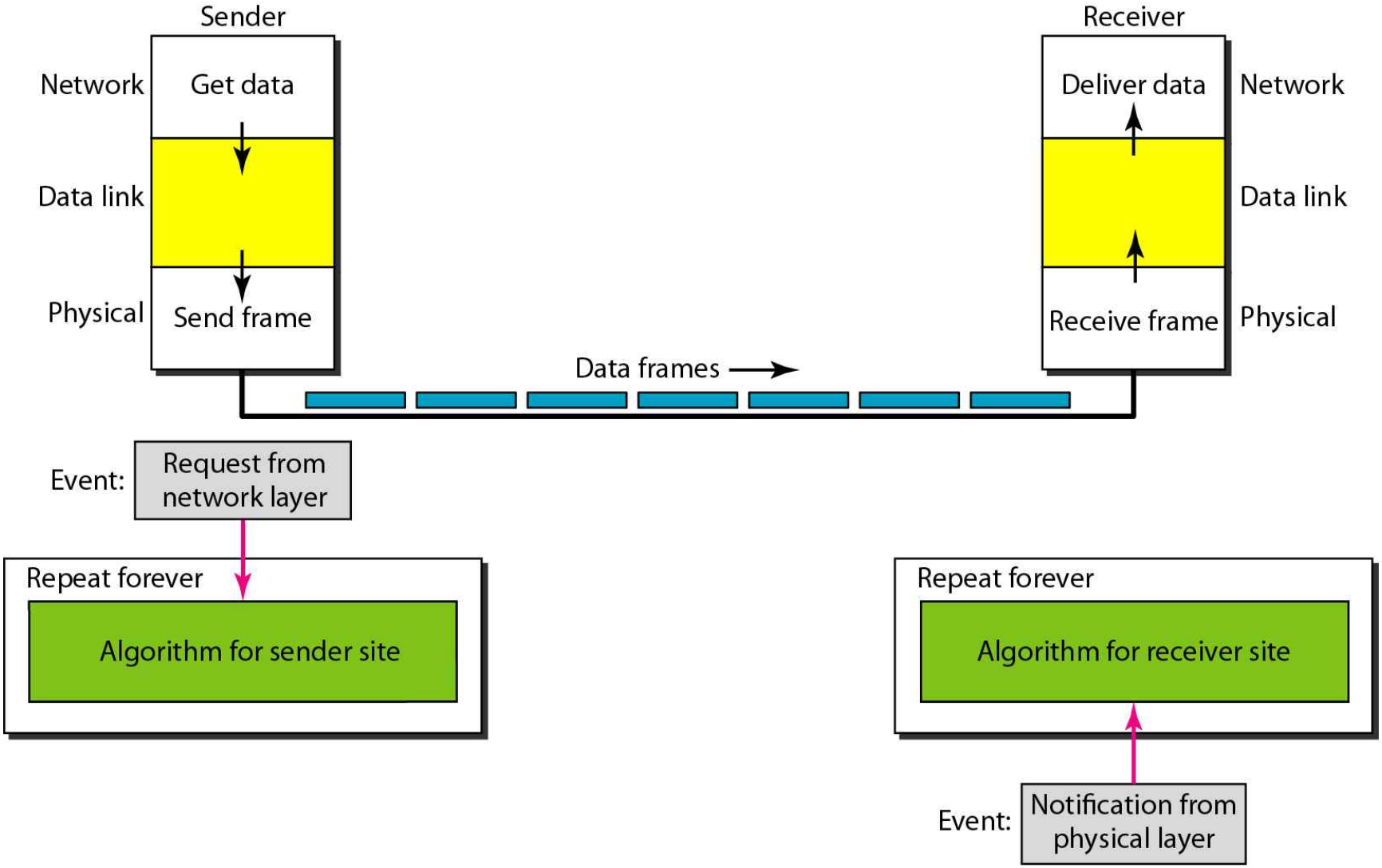
Simplest Protocol

- No need for flow or error control.
- Assumption
  - The receiver can immediately handle any frame without delay.
  - Unidirectional protocol in which data frames are traveling in only one direction.

(1) Design

- We need to introduce the idea of **events** in the protocol.
- At the sender, the procedure is constantly running; no action until there is a request from the network layer.
- At the receiver, the procedure is also constantly running; no action until notification from the physical layer

Figure 11.6 *The design of the simplest protocol with no flow or error control*



(2) Algorithms

Algorithm 11.1 *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 11.2 *Receiver-site algorithm for the simplest protocol*

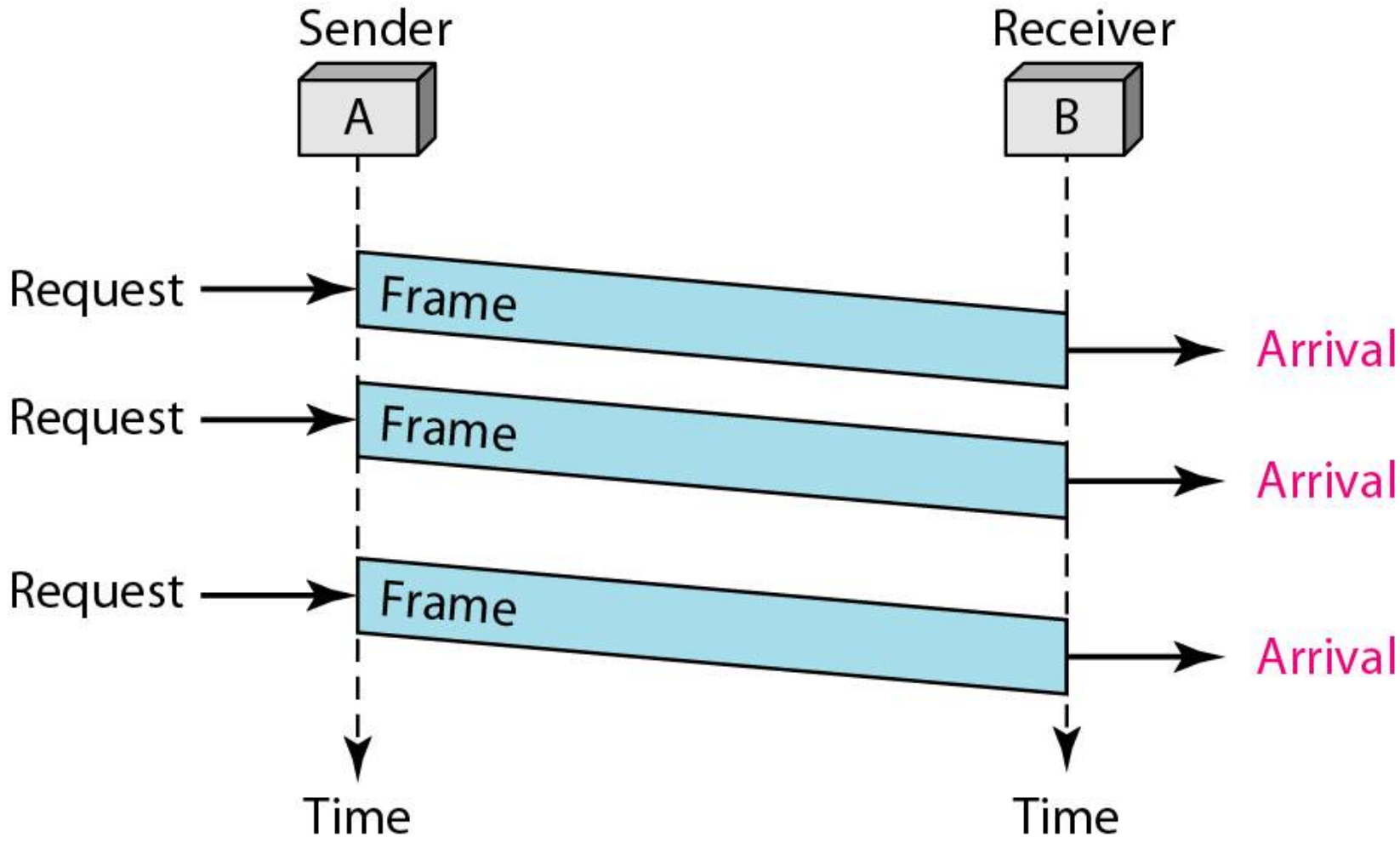
```
 1  while(true)                          // Repeat forever
 2  {
 3    WaitForEvent();                    // Sleep until an event occurs
 4    if(Event(ArrivalNotification))     //Data frame arrived
 5    {
 6       ReceiveFrame();
 7       ExtractData();
 8       DeliverData();                 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

Example 11.1

Figure 11.7 shows an example of communication using this protocol. It is very simple. 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.

Figure 11.7 *Flow diagram for Example 11.1*



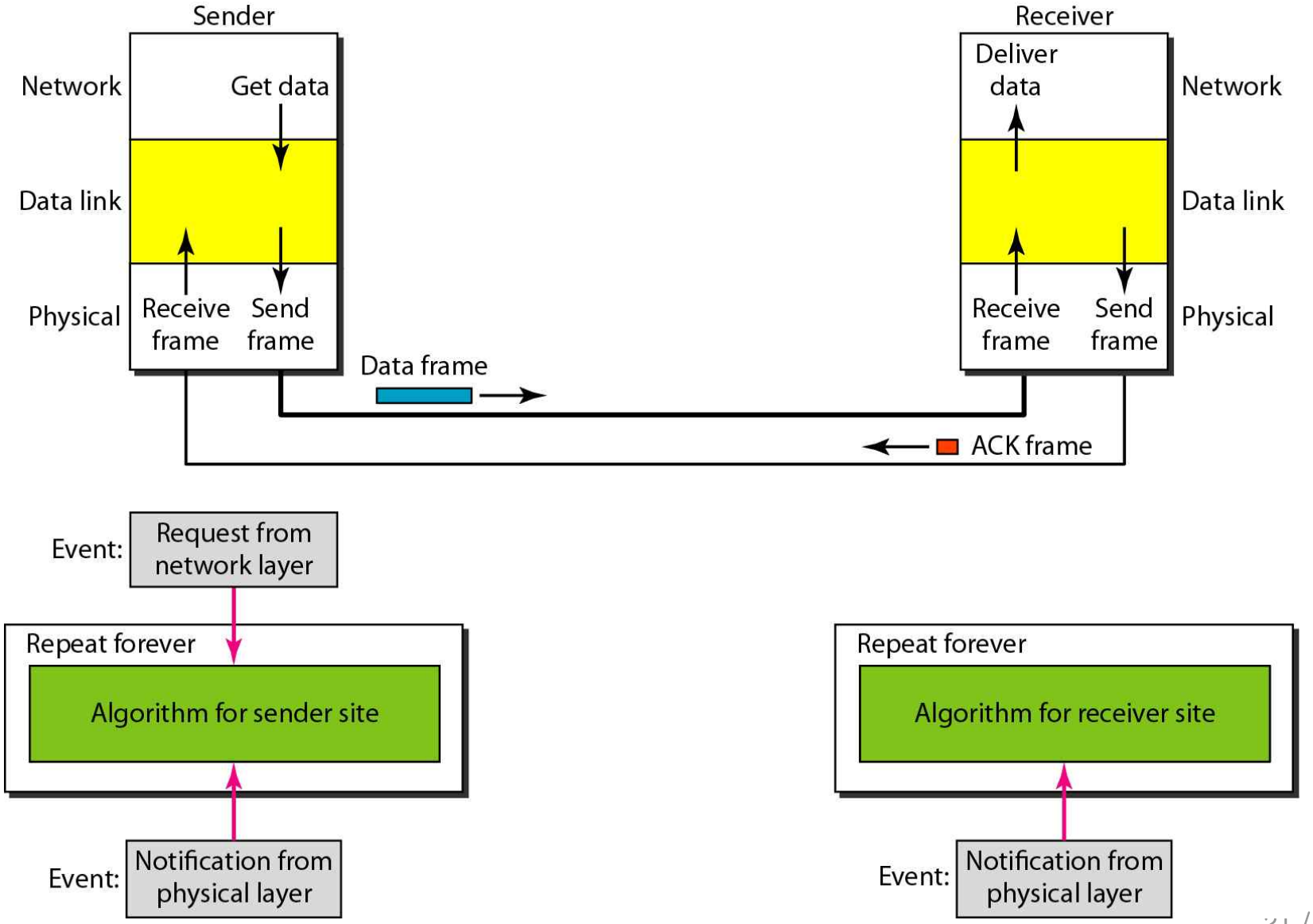
Stop-and-Wait Protocol

- Stop-and-Wait Protocol
  - The sender sends one frame, stops until it receives confirmation from the receiver (okay to sender ahead), and then sends the next frame.
- Assumption
  - Unidirectional communication for data frames,
  - but auxiliary ACK frames travel from the other direction.

(1) Design

- At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel. → a half-duplex link.

Figure 11.8 *Design of Stop-and-Wait Protocol*



(2) Algorithms

Algorithm 11.3 *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 }
```

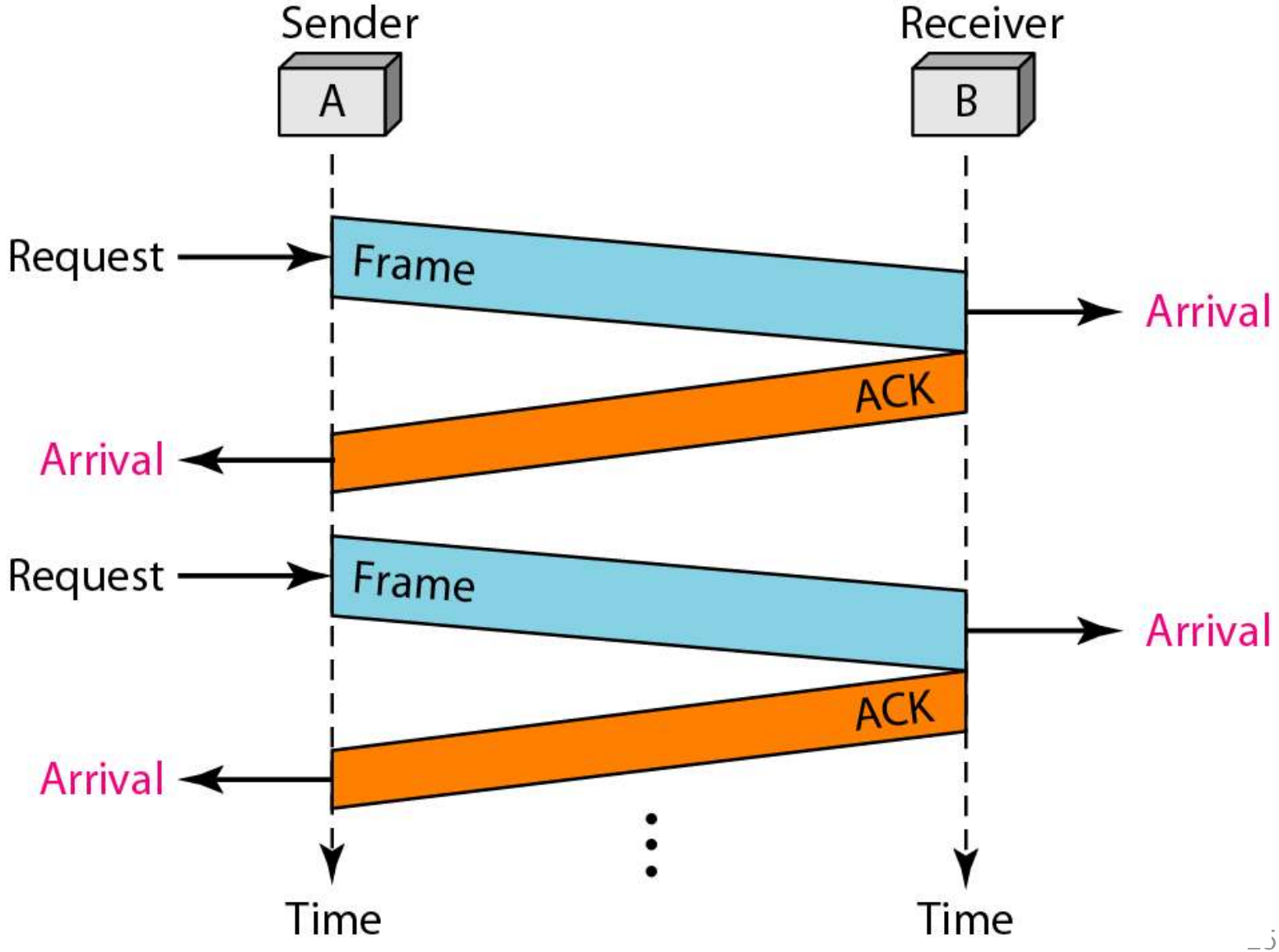
Algorithm 11.4 *Receiver-site algorithm for Stop-and-Wait Protocol*

```
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 11.2

Figure 11.9 shows an example of communication using this protocol. It is still very simple.
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.

Figure 11.9 *Flow diagram for Example 11.2*



5. Noisy Channels

Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent. We discuss three protocols in this section that use error control.

Topics discussed in this section:

Stop-and-Wait Automatic Repeat Request
Go-Back-N Automatic Repeat Request
Selective Repeat Automatic Repeat Request

Stop-and-Wait ARQ

- If the receiver does not respond when there is an error, how can the sender know which frame to resend?
- To remedy this problem,

> 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.

- Sequence Numbers (SN)
  - If the field is m bits long, the sequence numbers starts from 0, go to 2^m-1, and then are repeated.
  1. If the sender receives a ACK from the receiver, the sender sends the next frame numbered x+1.
  2. If the ACK is corrupted or lost, the sender resends the frame numbered x. → Duplicated but the receiver recognize this fact because of the SN.
  3. If the frame is corrupted or lost, the sender resends the frame numbered x after the time-out.

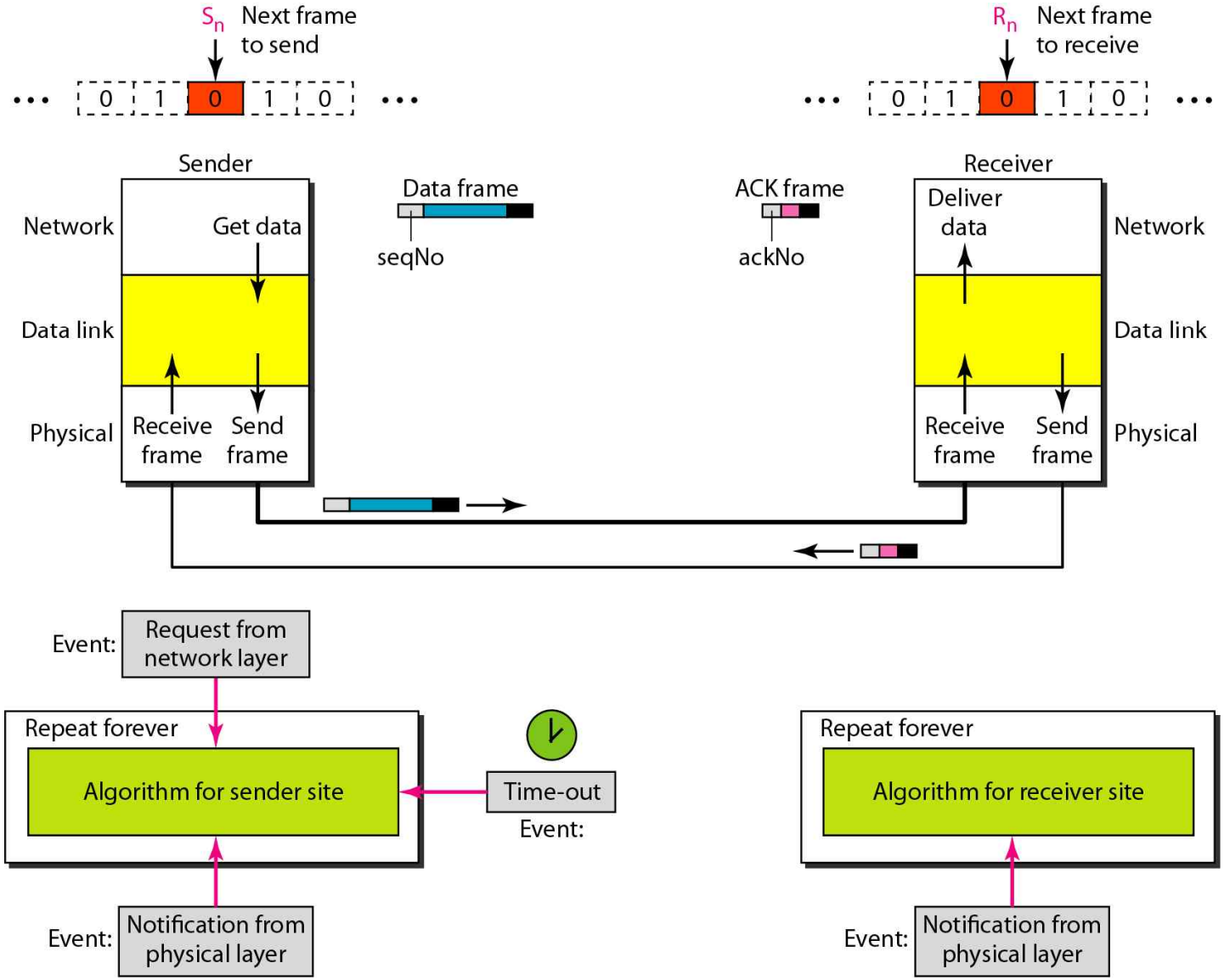
> In Stop-and-Wait ARQ, we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.

5. Noisy Channels

- Acknowledgment Numbers
  - If frame 1 has arrived safe and sound, the receiver sends an ACK frame with acknowledgement 0 (meaning frame 0 is expected).

In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Figure 11.10 *Design of the Stop-and-Wait ARQ Protocol*



Algorithm 11.5 *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
```

Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ* (continued)

```
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 11.6 *Receiver-site algorithm for Stop-and-Wait ARQ Protocol*

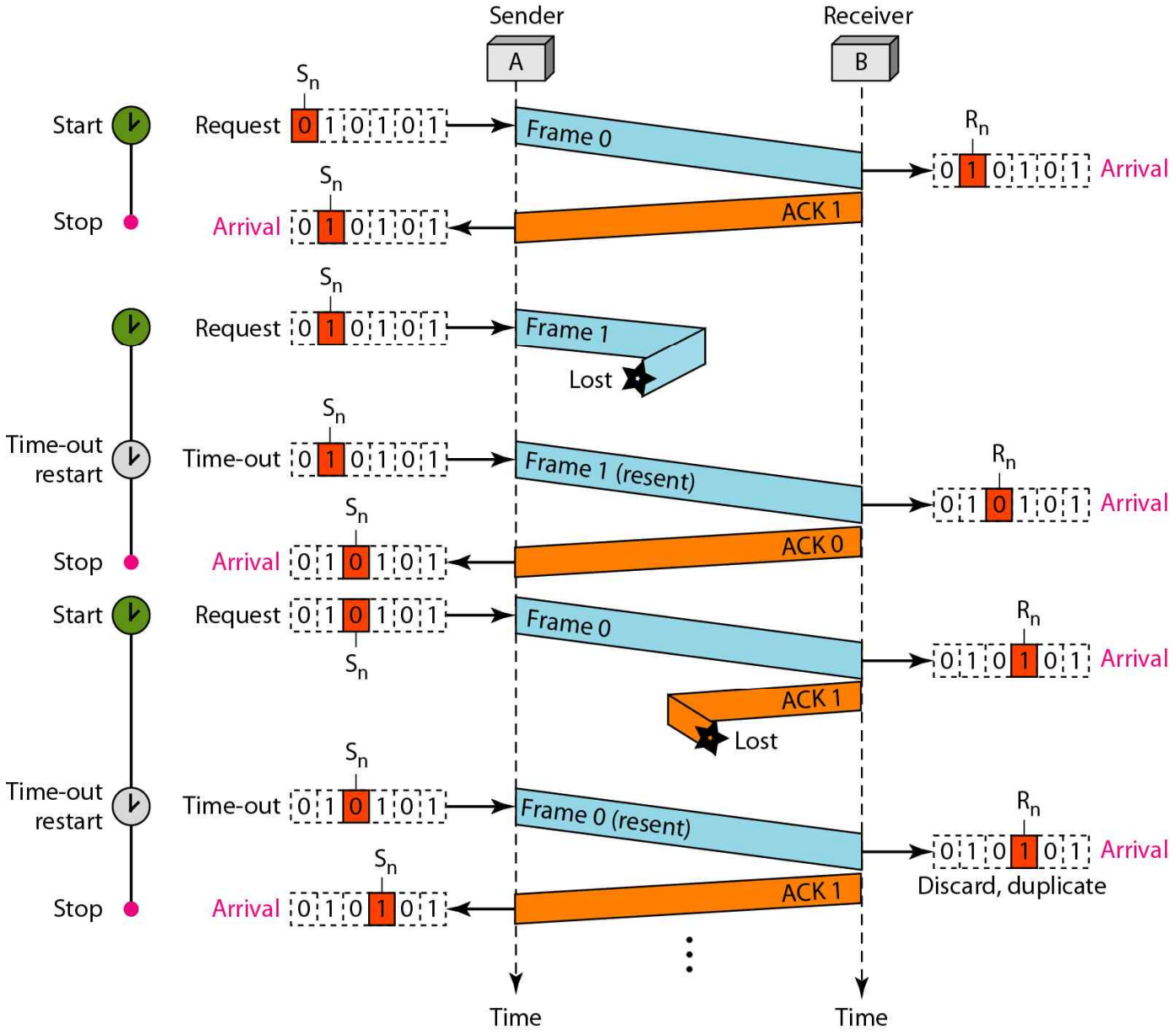
```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7       ReceiveFrame();
8       if(corrupted(frame));
9          sleep();
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 }
```

Example 11.3

Figure 11.11 shows an example of Stop-and-Wait ARQ.

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.

Figure 11.11 *Flow diagram for Example 11.3*



Example 11.4

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

1) The bandwidth-delay product is

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}$$

2) 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.

Example 11.5

What is the utilization percentage of the link in Example 11.4 if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?

Solution

The bandwidth-delay product is still 20,000 bits.

The system can send up to 15 frames or 15,000 bits during a round trip.

This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged frames, the utilization percentage is much less because frames have to be resent.