Data Link Control

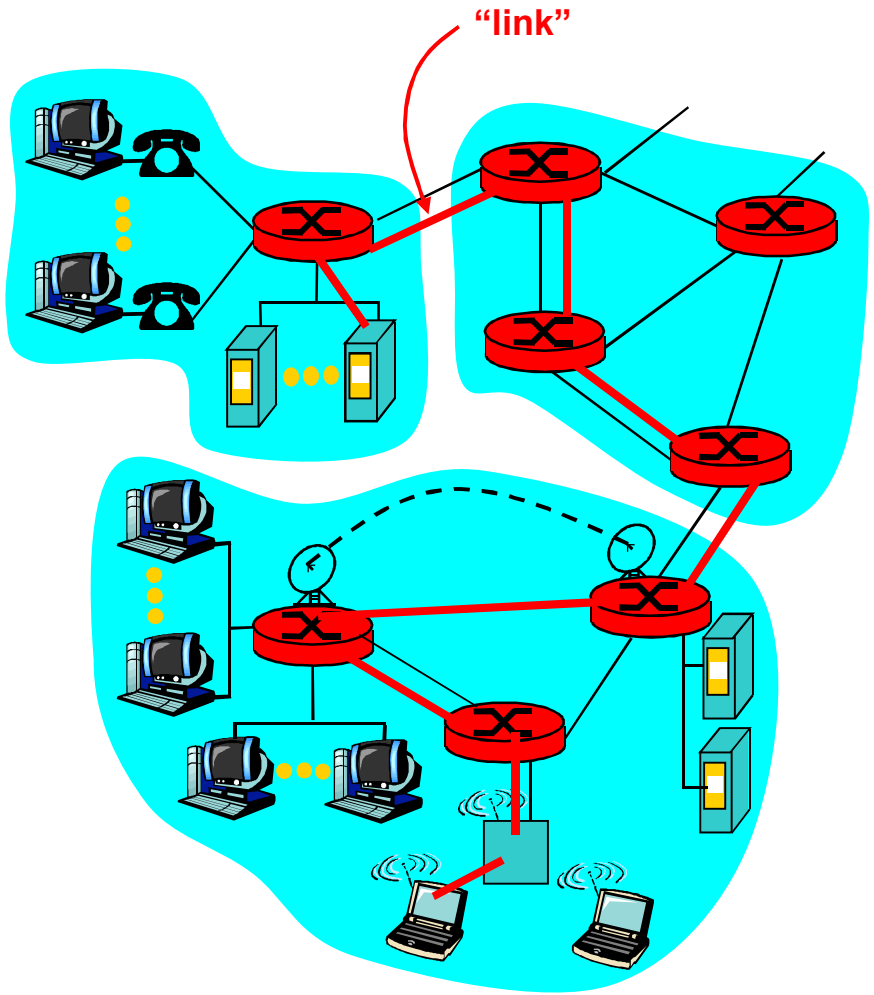
Flow Control : General view

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

Recall

Some terminology:

- hosts and routers are **nodes** (bridges and switches too)
- communication channels that connect adjacent nodes along communication path are **links**
- 2-PDU is a **frame**



data-link layer has responsibility of transferring frame from one node to adjacent node over a link

Flow and Error Control

- Flow Control
  - Flow control refers to a set of procedures used to **restrict the amount of data that the sender can send** before waiting for acknowledgment from the receiver
    - Stop-and-Wait flow control
    - Sliding-Window flow control
- Error Control: based on automatic repeat request, which is the retransmission of data
  - Refers to procedures to **detect and correct** errors
  - Includes the following actions:
    - Error detection
    - Positive Acknowledgement (**ACK**): if the frame arrived with no errors
    - Negative Acknowledgement (**NAK**): if the frame arrived with errors
    - Retransmissions after **timeou**t: Frame is retransmitted after certain amount of time if no acknowledgement was received
  - These actions are called A**utomatic Repeat Request (ARQ)**

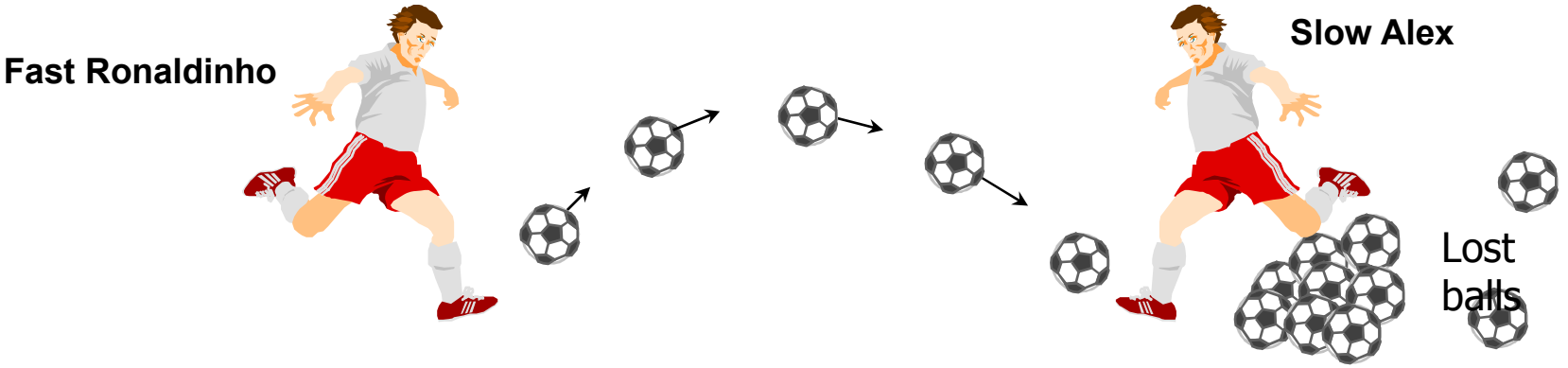
Flow and Error Control

- Usually Error and flow control protocols are combined together to provide reliable data transfer service called data link control
  - Stop-and-Wait ARQ
  - Go-Back-N ARQ
  - Selective repeat ARQ
- ARQ provide **reliable data transfer** service over **unreliable networks**
- ARQ ensure that transmitted **data** is delivered accurately to the destination despite errors that occur during transmission and satisfies the following:
  - **Error free**
  - **Without duplicates**
  - **Same order** in which they are transmitted
  - **No loss**

Main focus of Flow Control

- Ensuring the sending entity does not overwhelm the receiving entity
  - Preventing buffer overflow at the receiver
- It is to prevent the receiver from getting overloaded by the data sent by the faster-transmitting sender.
  - **If a sender is on a powerful machine and it is transmitting the data at the faster rate,**
    - **even though the data transmitted is error free,**
  - **it may happen that the receiver on slower-end is unable to receive data at that speed**
  - **it may loose some data.**
- There are two methods of flow control, feedback-based flow control and rate-based flow control

Flow Control



- What to do with a sender that wants to transmit frames faster than the receiver can accept them ???
- Even if transmission is error free, the receiver may be unable to handle the frames as they arrive and lose
  - Might be possible for the sender to simply insert a delay to slow down sufficiently to keep from swamping the receiver
- Two approaches for flow control
  - **Feedback-based flow control:** the receiver sends back information to the sender giving it permission to send more data or at least telling the sender how the receiver is doing.
  - **Rate-based flow control: the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.**

Feedback-based control

Here, after the receiver receives the first frame:

- **it informs the sender,**
- **permits it to send more information**
- **it also inform about the status of the receiver**

There are two protocols of feedback-based flow control:

- sliding window protocol
- and stop-and-wait protocol

Rate-based control

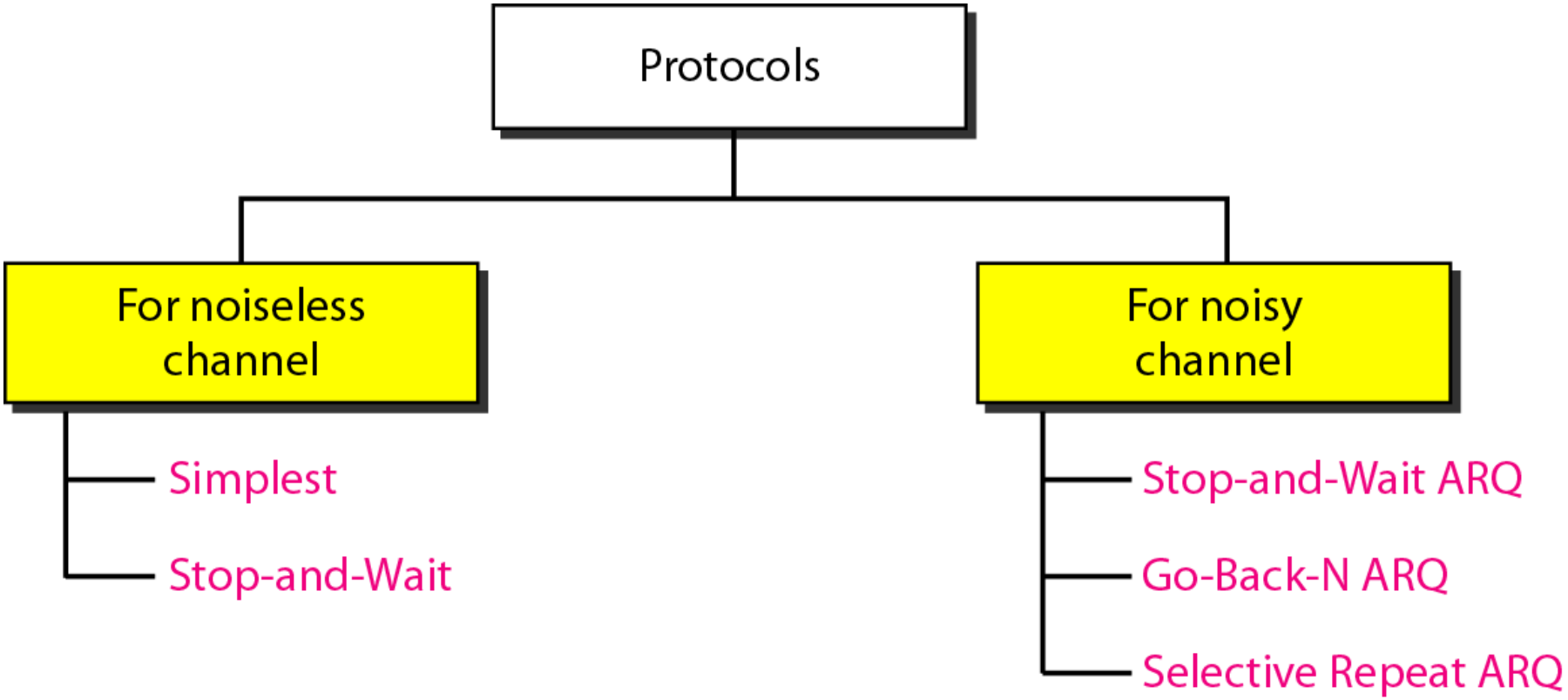
Here, when a sender transmits the data at a faster rate to the receiver receiver is unable to receive the data at that speed,

- then the built-in mechanism in the protocol will **limit the rate of transmission at which the sender is transmitting data without any feedback from the receiver**.

Flow Control : Specific Protocols

- Chanel: simplex or duplex
- Buffer capacity at receiver: limited or infinite
- Chanel: Ideal or not:
  - Error free transmission, *no frames are lost, duplicated, or corrupted*
  - Noisy chanel
- Network layer at the sender's end is always ready with data

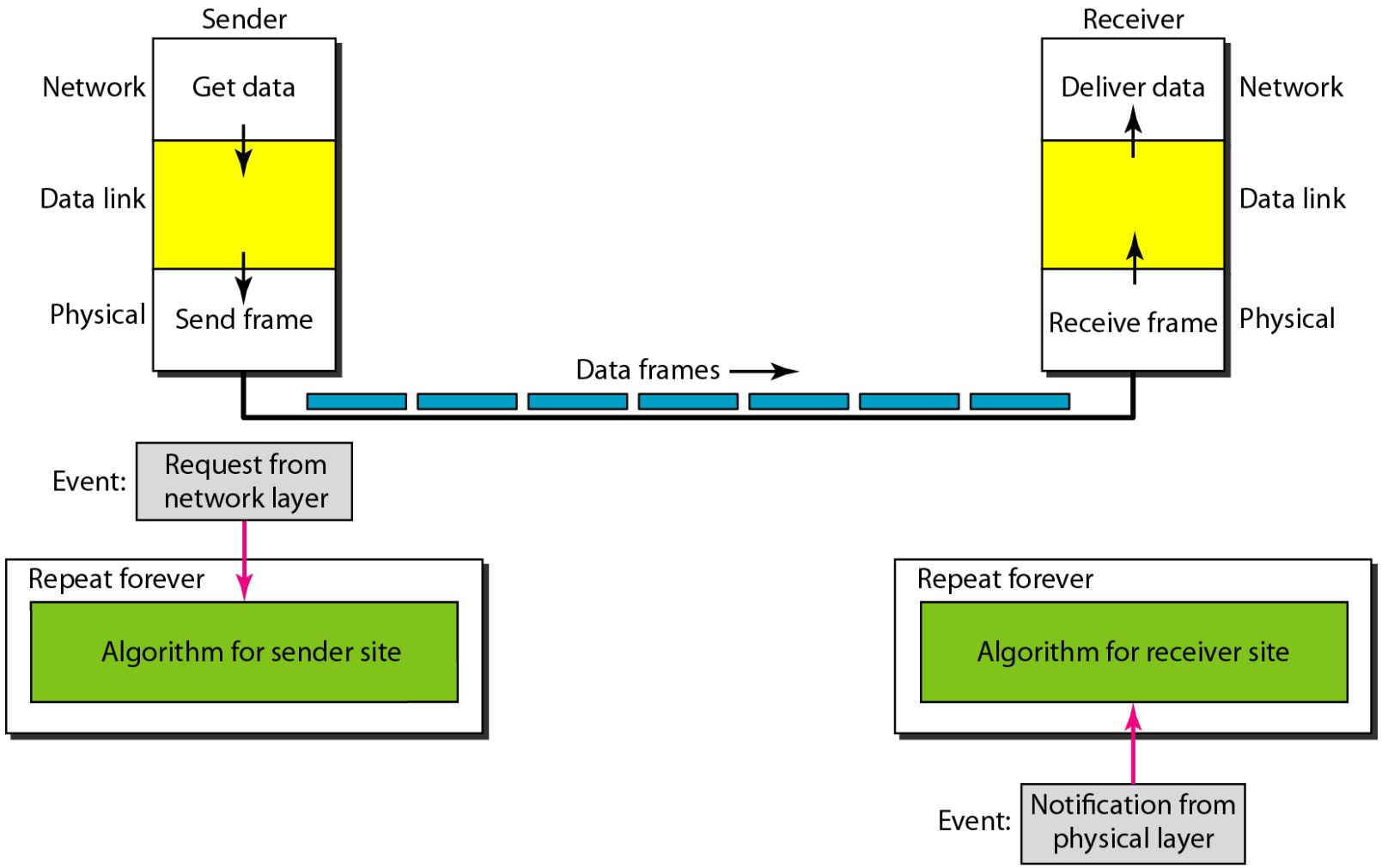
Figure 11.5 *Taxonomy of protocols of this chapter*



Flow Control: Initial Assumptions

- Simplex Channel
- Infinite buffer capacity with the receiver
- Error free transmission
- *ideal channel in which no frames are lost, duplicated, or corrupted*
- No need for flow control

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



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 }
```

Protocol Definitions

```
#define MAX_PKT 1024                                  /* determines packet size in bytes */

typedef enum {false, true} boolean;                   /* boolean type */
typedef unsigned int seq_nr;                          /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;             /* frame_kind definition */

typedef struct {                                      /* frames are transported in this layer */
  frame_kind kind;                                    /* what kind of a frame is it? */
  seq_nr seq;                                         /* sequence number */
  seq_nr ack;                                         /* acknowledgement number */
  packet info;                                        /* the network layer packet */
} frame;
```

Continued →

Some definitions needed in the protocols to follow.
These are located in the file protocol.h.

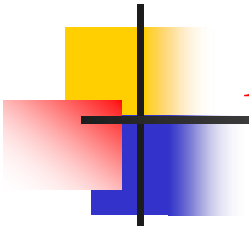
Protocol Definitions (ctd.)

Some definitions needed in the protocols to follow. These are located in the file protocol.h.

```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Unrestricted Simplex Protocol

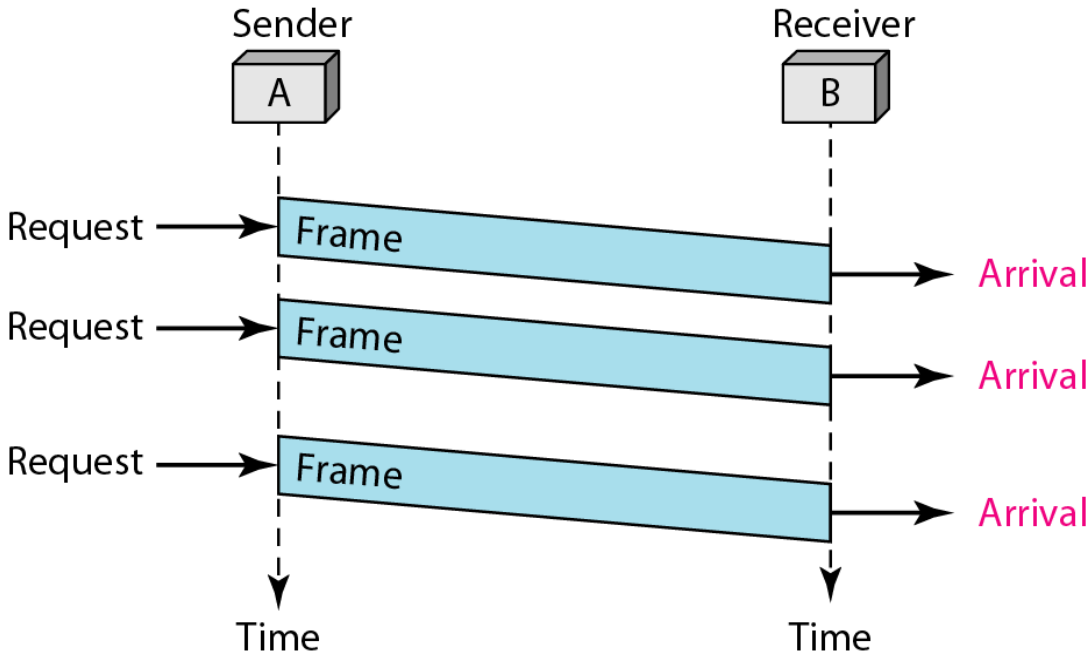
```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free,
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

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

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

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

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

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



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.

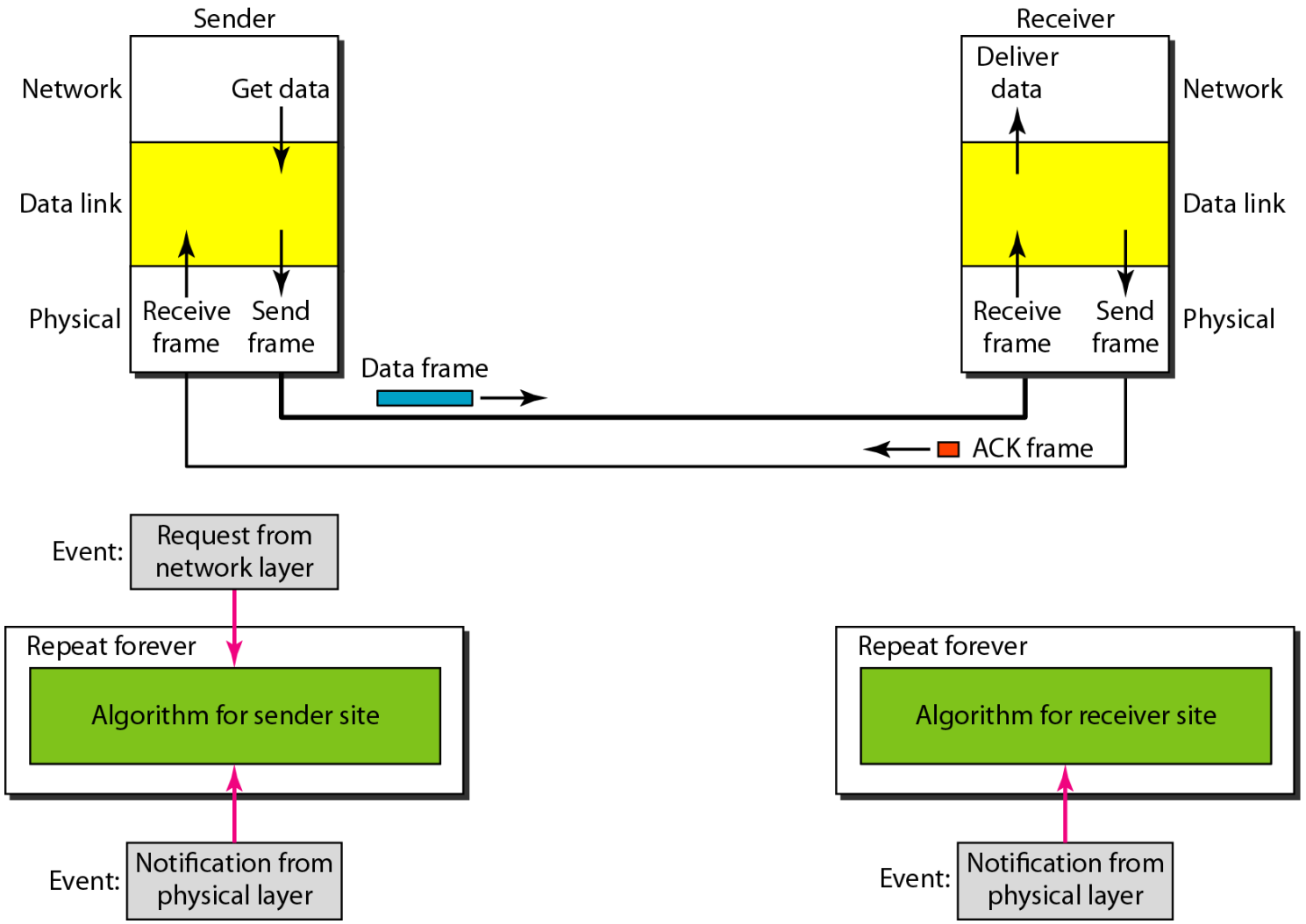
Figure 11.7 *Flow diagram for Example 11.1*



Some assumptions dropped

- Infinite capacity in the buffer of the receiver.
- Need for “flow control”
- Stop-n-Wait protocol
  - Sender sends a frame
  - And waits for a signal in the form of a dummy frame
  - No seq no. is required since the line is still error free

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



Algorithm 11.3 *Sender-site algorithm for Stop-and-Wait Protocol*

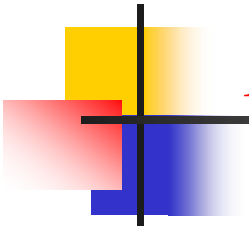
```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       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 }
```

Simplex Stop-and-Wait Protocol

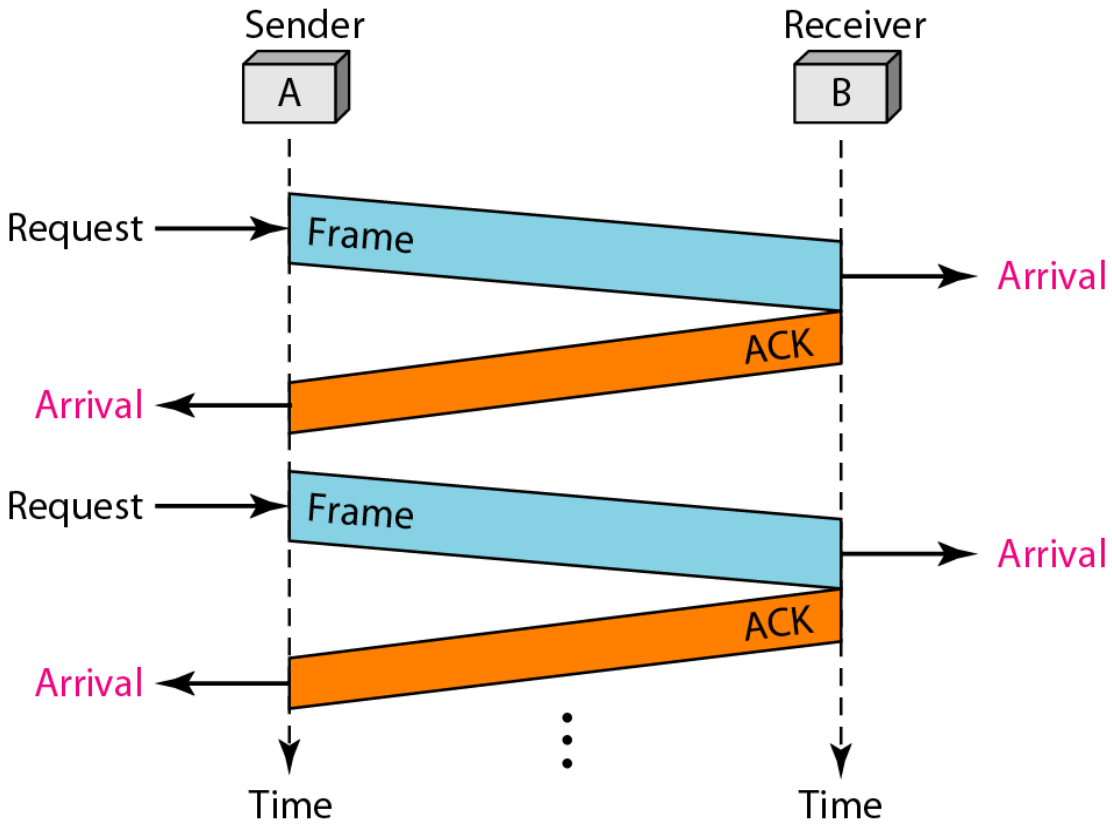
```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

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

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

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

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



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*



Flow Control for 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.*
- *3 protocols that use error control*
  - Stop-and-Wait Automatic Repeat Request
    Go-Back-N Automatic Repeat Request
    Selective Repeat Automatic Repeat Request
- Transmission time (t_{frame})
  - Time taken to emit all bits into medium
- Propagation time (t_{prop})
  - Time for a bit to traverse the link

Stop and Wait Operation

- Simple
- Inefficient