CAP256: Computer Network
Data Link Layer – Flow Control

Dr. Manmohan Sharma
School of Computer Applications
Lovely Professional University

Flow and Error Control

- Data-link layer is responsible for implementation of point-to-point flow and error control mechanism.
- When a data frame (Layer-2 data) is sent from one host to another over a single medium, it is required that the sender and receiver should work at the same speed.
  - That is, sender sends at a speed on which the receiver can process and accept the data.
- What if the speed (hardware/software) of the sender or receiver differs?
  - If sender is sending too fast the receiver may be overloaded, (swamped) and data may be lost.

Flow Control

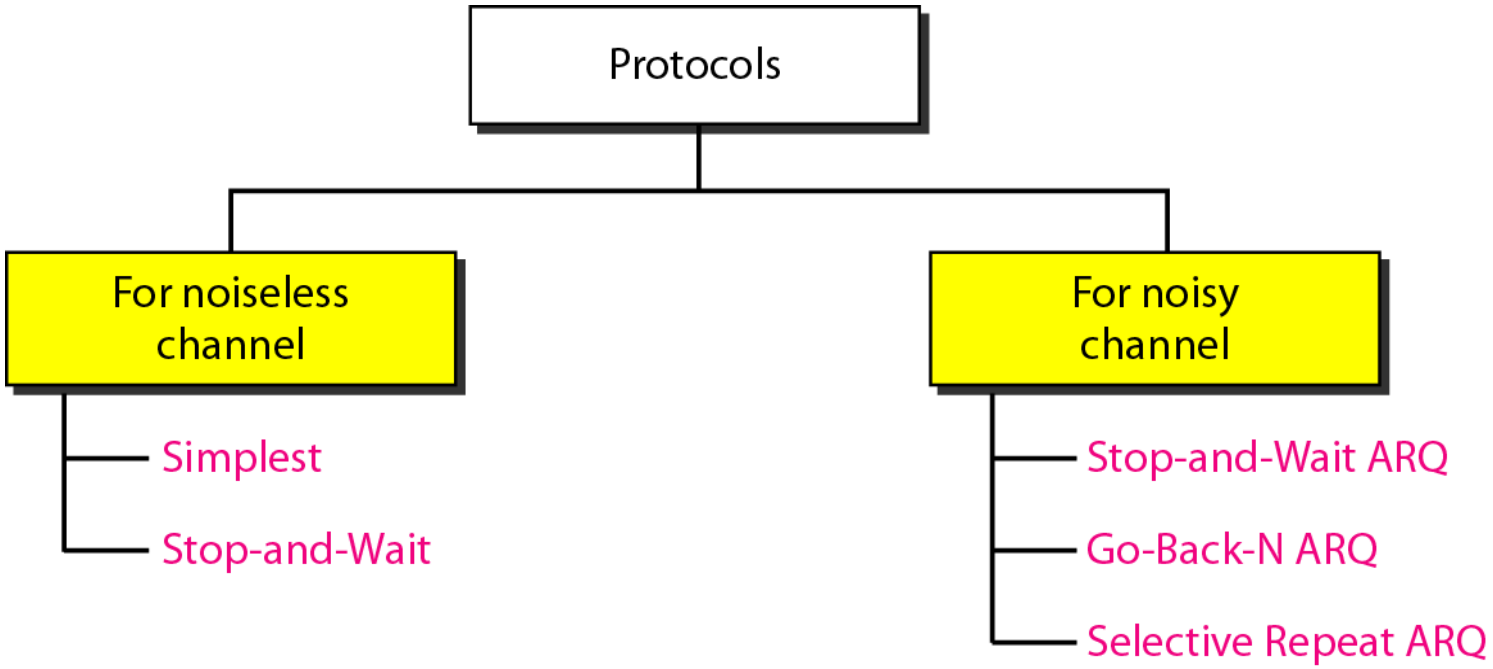
- **Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.**
- Two types of mechanisms can be deployed to control the flow:
  - **Stop and Wait:** This flow control mechanism forces the sender after transmitting a data frame to stop and wait until the acknowledgement of the data-frame sent is received.
  - **Sliding Window:** In this flow control mechanism, both sender and receiver agree on the number of data-frames after which the acknowledgement should be sent. As we learnt, stop and wait flow control mechanism wastes resources, this protocol tries to make use of underlying resources as much as possible.

Error Control

- When data-frame is transmitted, there is a probability that data-frame may be lost in the transit or it is received corrupted.
- In both cases, the receiver does not receive the correct data-frame and sender does not know anything about any loss. In such case, both sender and receiver are equipped with some protocols which helps them to detect transit errors such as loss of data-frame.
- Hence, either the sender retransmits the data-frame or the receiver may request to resend the previous data-frame.
- Requirements for error control mechanism:
  - **Error detection:** The sender and receiver, either both or any, must ascertain that there is some error in the transit.
  - **Positive ACK** : When the receiver receives a correct frame, it should acknowledge it.
  - **Negative ACK:** When the receiver receives a damaged frame or a duplicate frame, it sends a NACK back to the sender and the sender must retransmit the correct frame.
  - **Retransmission:** The sender maintains a clock and sets a timeout period. If an acknowledgement of a data-frame previously transmitted does not arrive before the timeout the sender retransmits the frame, thinking that the frame or it's acknowledgement is lost in transit.

Protocols used for Flow Control

- 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

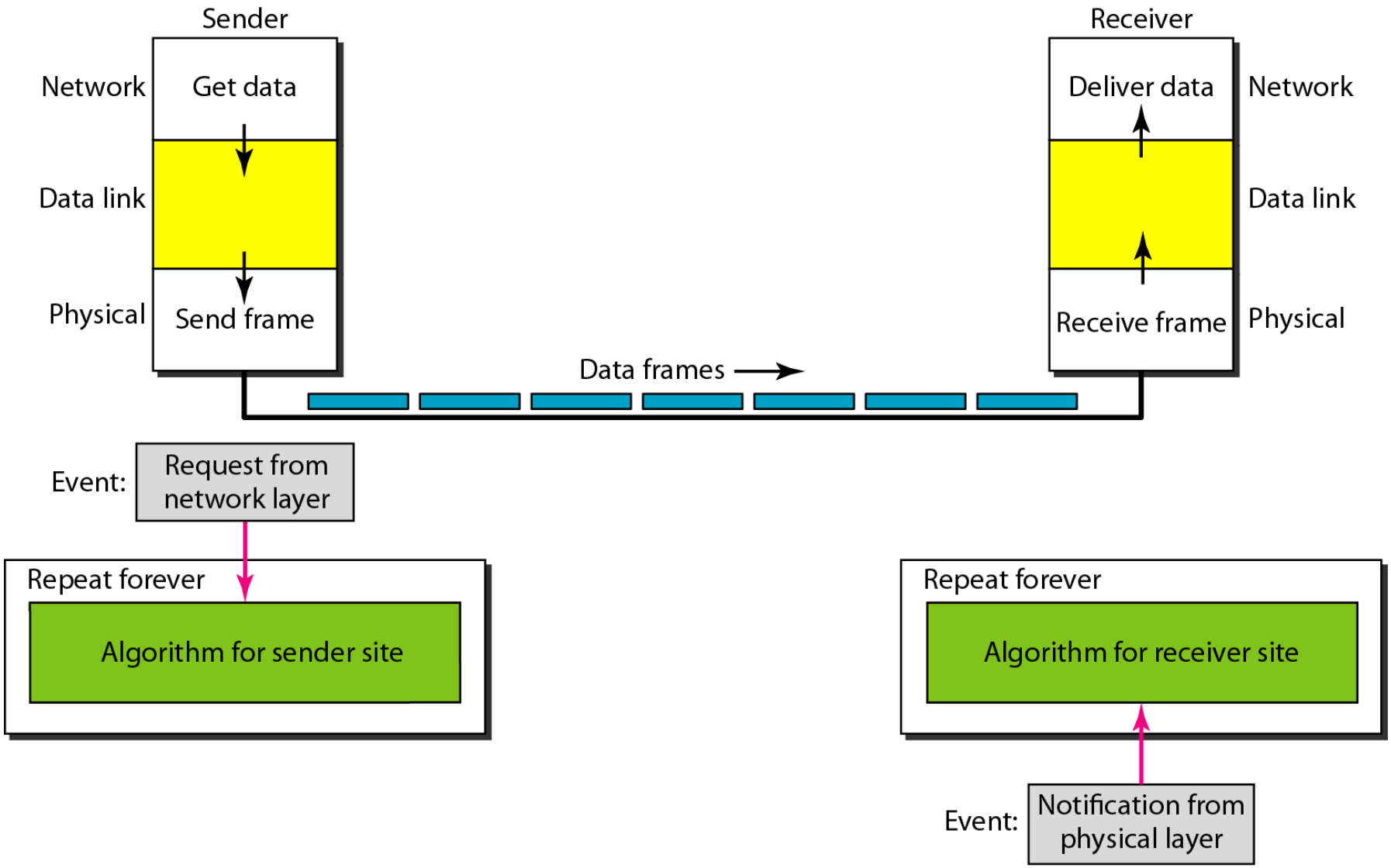
- Noiseless channel is an ideal channel in which no frames are lost, duplicated, or corrupted.
- Two protocols are available for this type of channel.
  - **Simplest Protocol:** This type of protocol has no flow or error control
  - **Stop-and-Wait Protocol:** In this type of protocol sender sends one frame, stops until it receives agree from receiver and then sends the next frame

Simplest Protocol

- Our first protocol, which we call the Simplest Protocol, is one that has no flow or error control. Like other protocols, it is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver.
- 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 network layer, which can also accept the packet immediately.
- In other words, the receiver can never be fill out with incoming frames.

Assumptions for Simplest Protocol

- The channel is a perfect noiseless channel.
- Hence an ideal channel in which no frames are lost, duplicated, or corrupted.
- No flow control and error control used.
- It is a unidirectional protocol in which data frames are traveling in only one direction- from the sender to receiver.
- Both transmitting and receiving network layer are always ready.
- Processing time that is small enough to be negligible.
- Infinite buffer space is available.


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

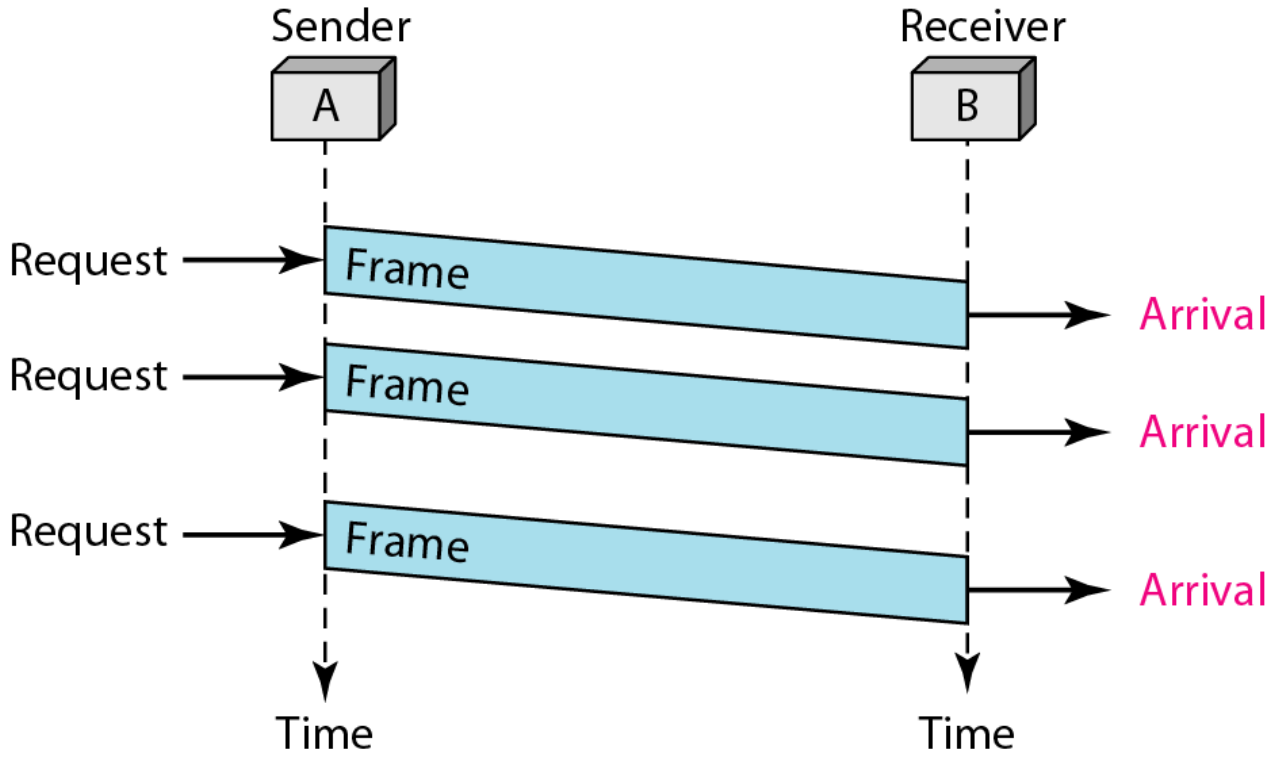
Sender-site algorithm for the simplest protocol

```
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 }
```

Receiver-site algorithm for the simplest protocol

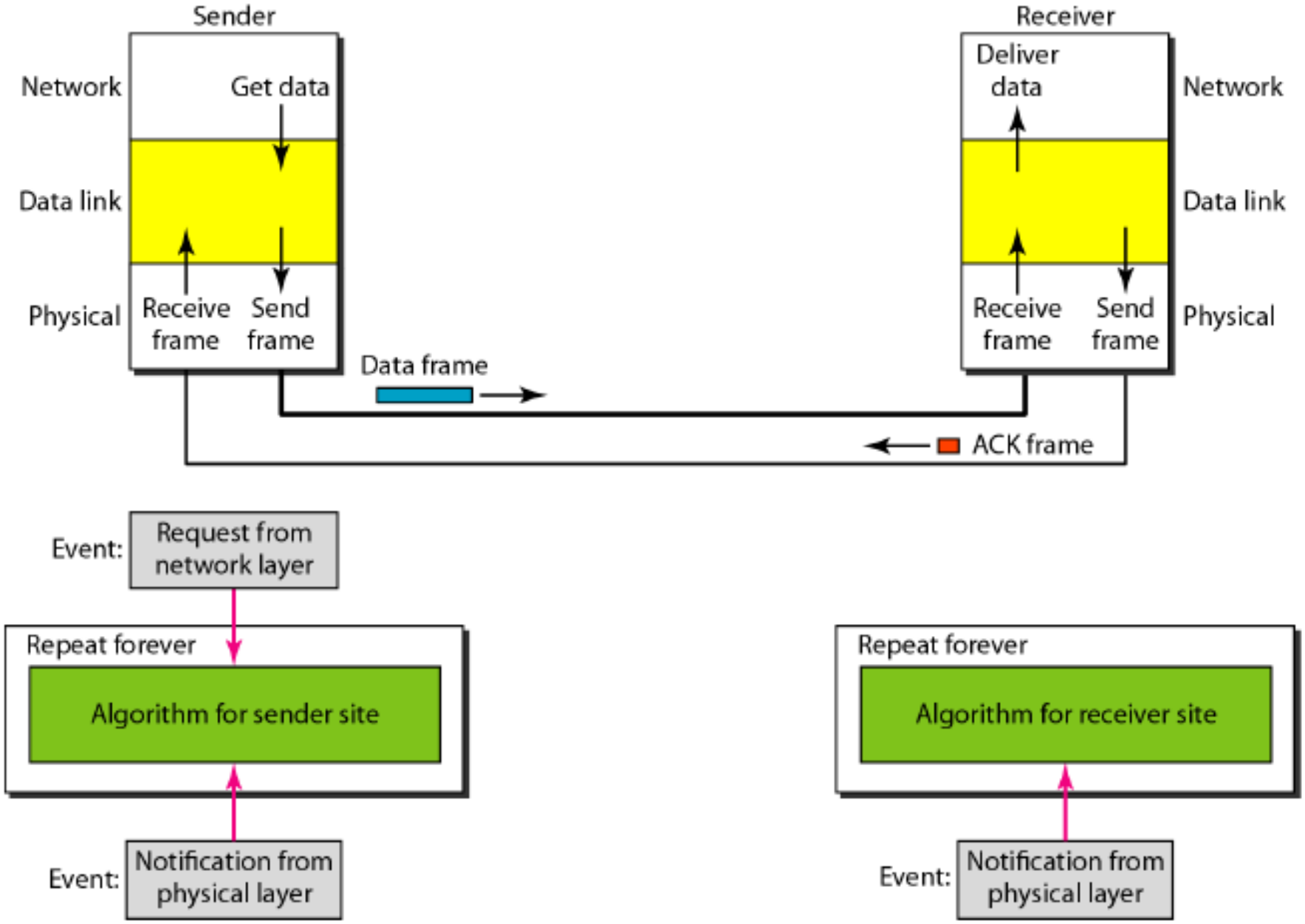
```
1  while(true)                              // Repeat forever
2  {
3    WaitForEvent();                        // Sleep until an event occurs
4    if(Event(ArrivalNotification))         //Data frame arrived
5    {
6       ReceiveFrame();
7       ExtractData();
8       DeliverData();             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

Following figure 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.



Stop and Wait Protocol

- This flow control mechanism forces the sender after transmitting a data frame to stop and wait until the acknowledgement of the data-frame sent is received.
- The following assumption has been made for developing the Stop-and-Wait Protocol
  - The channel is a perfect noiseless channel.
  - Flow control used
  - It is a bidirectional protocol in which frames are traveling in both direction
  - Both transmitting and receiving network layer are always not ready.
  - Processing time considerable
  - Finite buffer space is available
  - The receiver may not be always ready to receive the next frame (finite buffer storage).
  - Receiver sends a positive acknowledgment frame to sender to transmit the next data frame which showed in the below figure(3.4).
  - Error-free communication channel assumed. No retransmissions used.


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

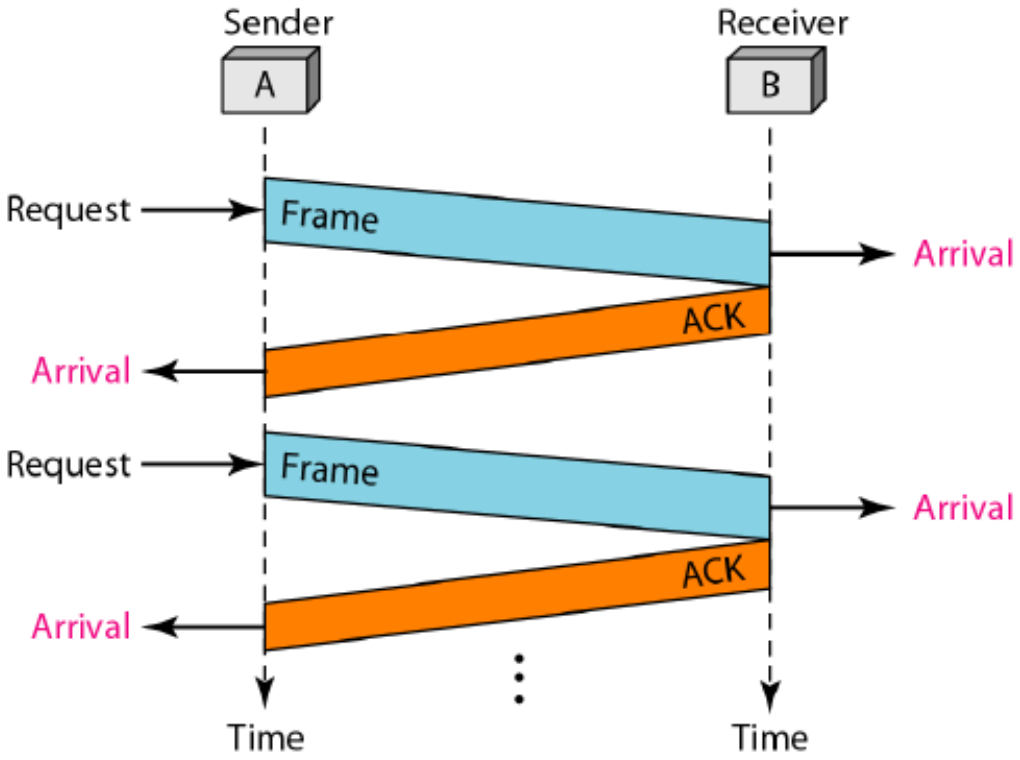
Sender-site algorithm for Stop-and-Wait Protocol

```
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 }
```

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 }
```

Following figure 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.

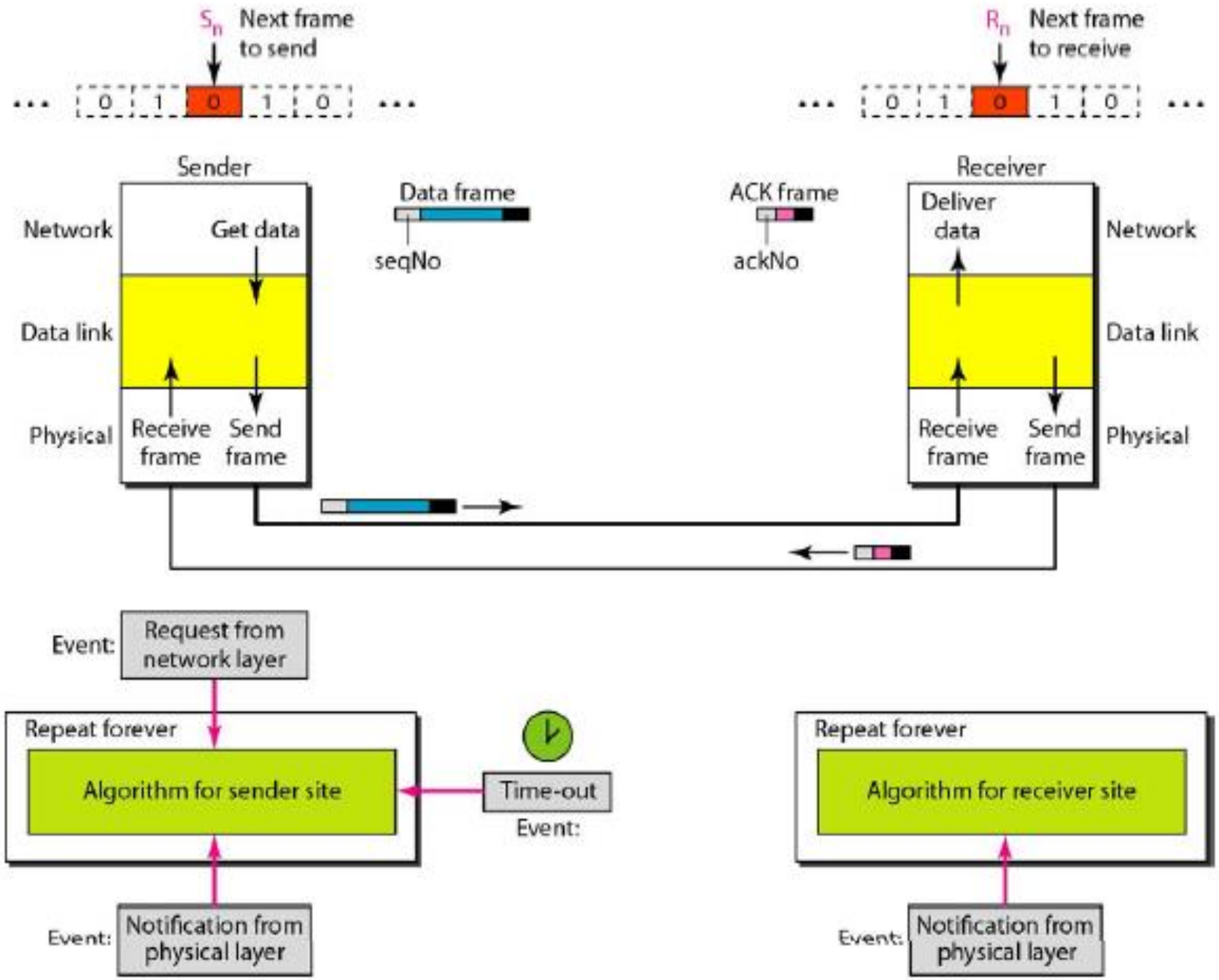


Noisy Channels

- Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are non-existent.
- Three protocols are discussed to use error control while transmission using noisy channels:
  - Stop-and-Wait Automatic Repeat Request
  - Go-Back-N Automatic Repeat Request
  - Selective Repeat Automatic Repeat Request

Stop-and-Wait ARQ

- Automatic Repeat Request (ARQ), an error control method, is incorporated with stop and wait flow control protocol.
- Error correction in Stop-and-Wait 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 ARQ, acknowledgment number always announces in modulo-2 arithmetic the sequence 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
Receiver
Data frame
seqNo
ACK frame
ackNo
Network
Get data
Data link
Physical
Receive frame
Send frame
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

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
```

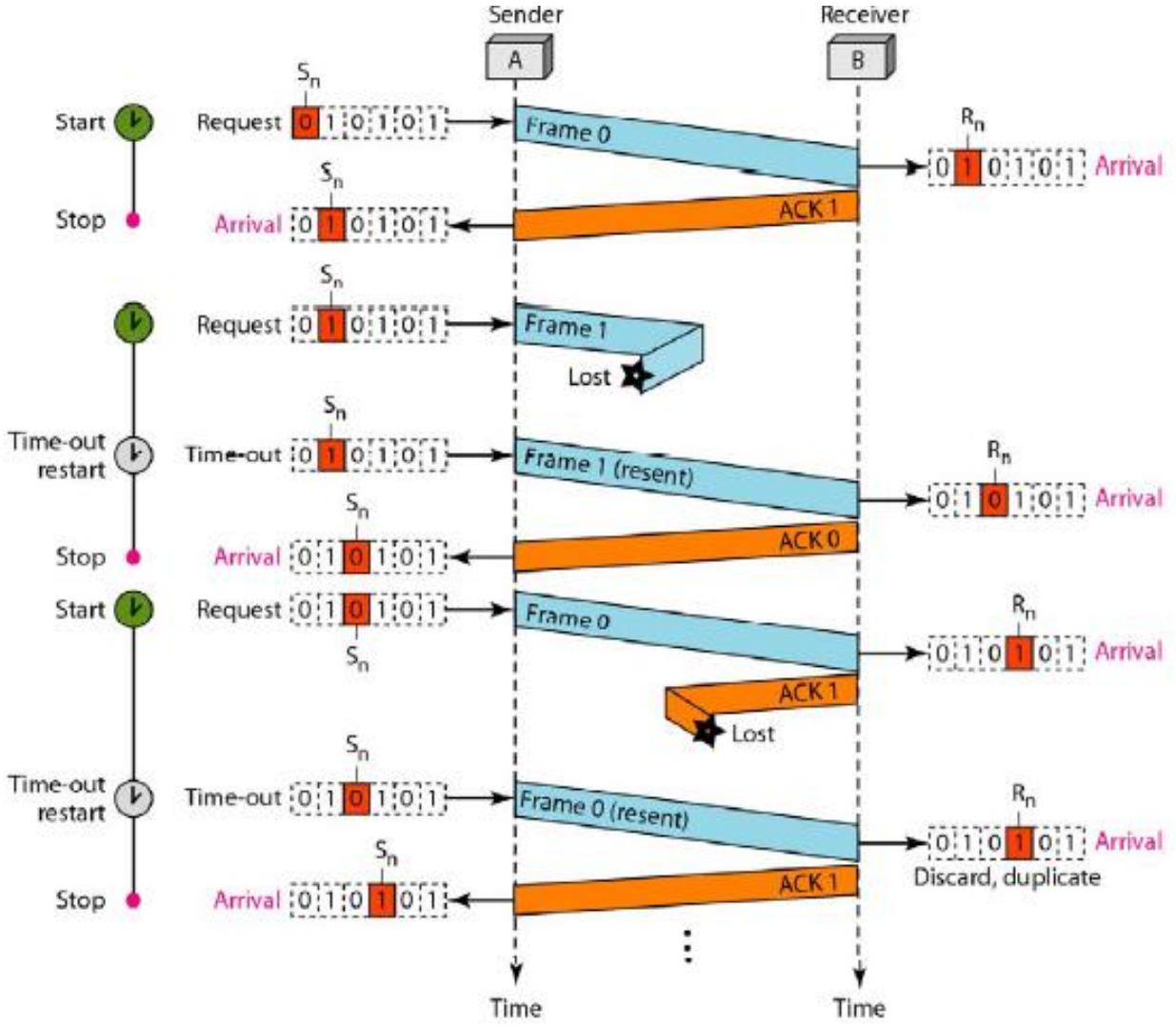
(continued)

```
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  }
```

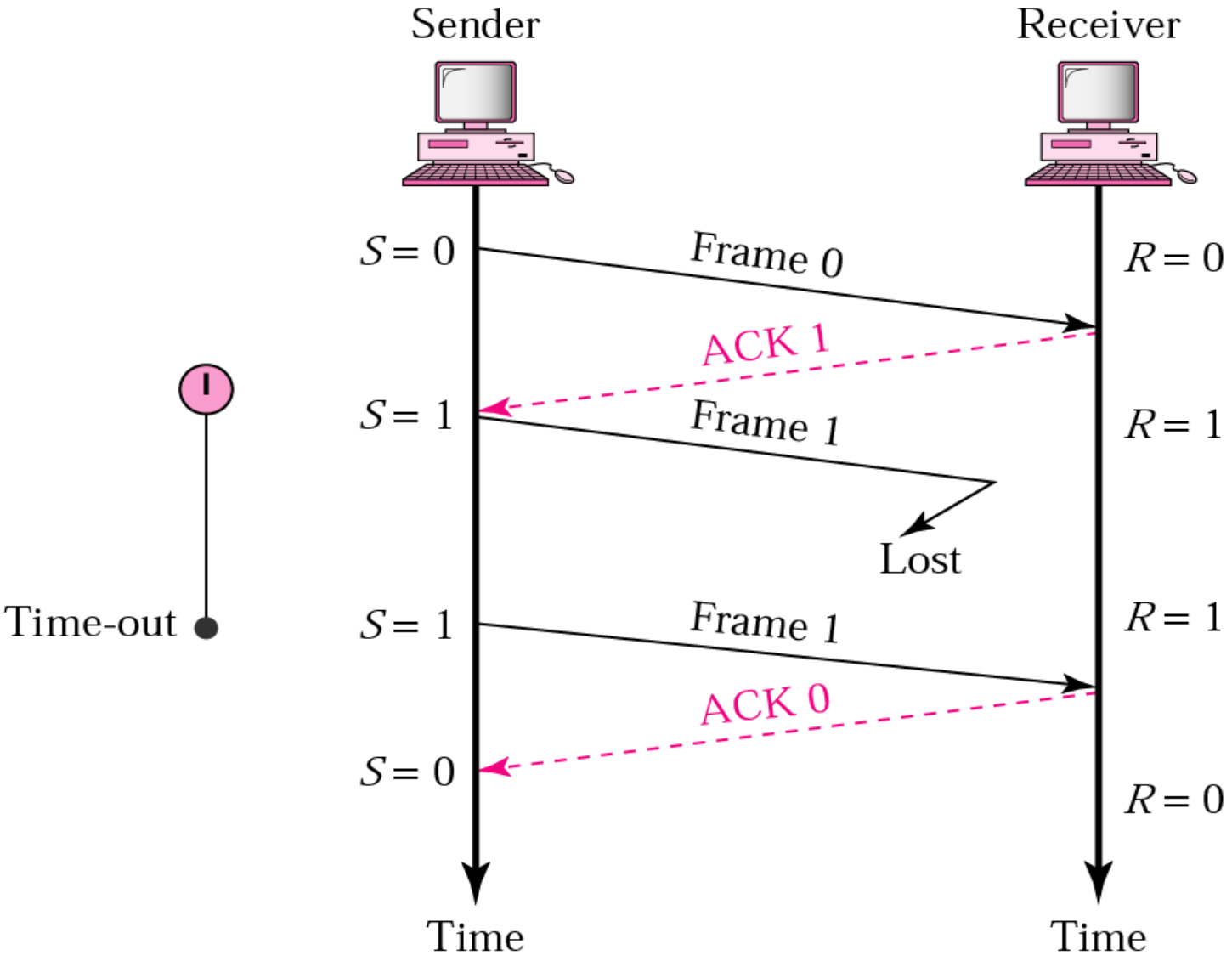
Receiver-site algorithm for Stop-and-Wait ARQ Protocol

```
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 }
```

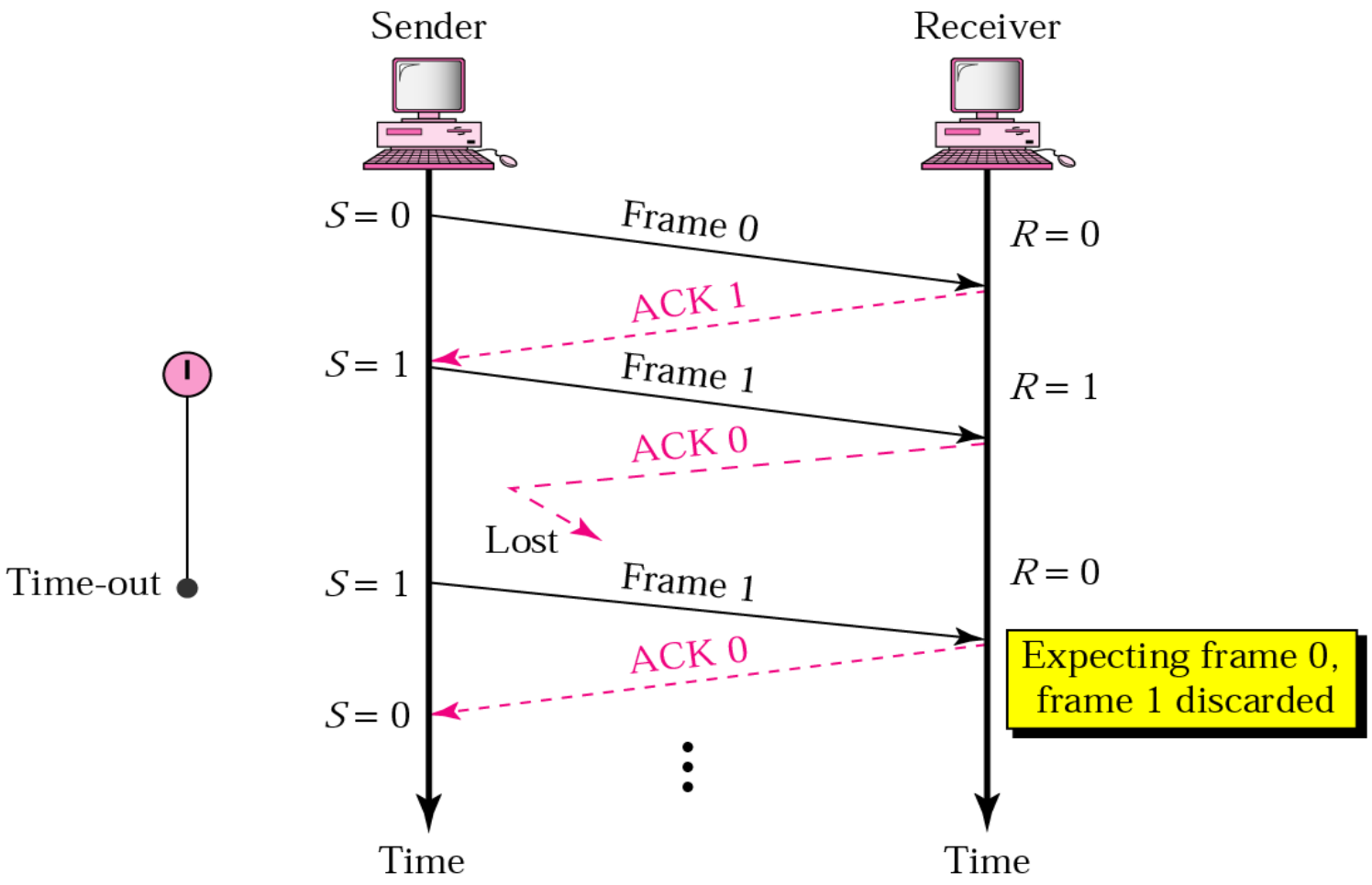
Following figure shows an example of Stop-and-Wait ARQ. Frame 0 is sent and acknowledged. Frame 1 is lost and resent after the time-out. The resent frame 1 is acknowledged and the timer stops. Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.



Stop-and-Wait ARQ, lost frame

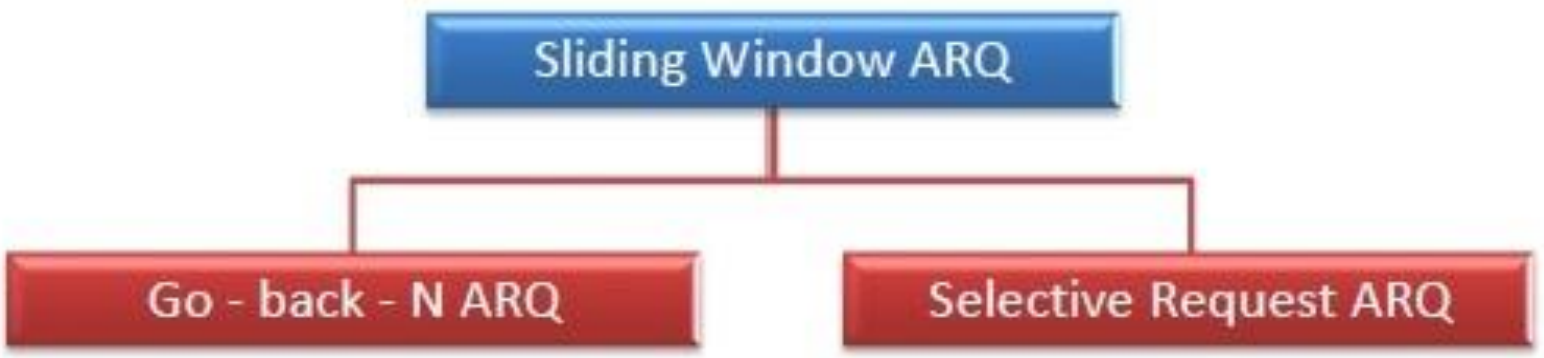


Stop-and-Wait ARQ, lost ACK frame



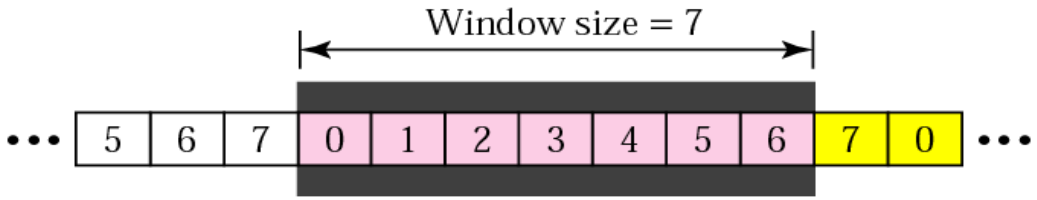
Sliding Window Protocols

- Sliding window protocol is a flow control protocol.
- It allows the sender to send multiple frames before needing the acknowledgements.
- Sender slides its window on receiving the acknowledgements for the sent frames.
- This allows the sender to send more frames.
- It is called so because it involves sliding of sender's window.

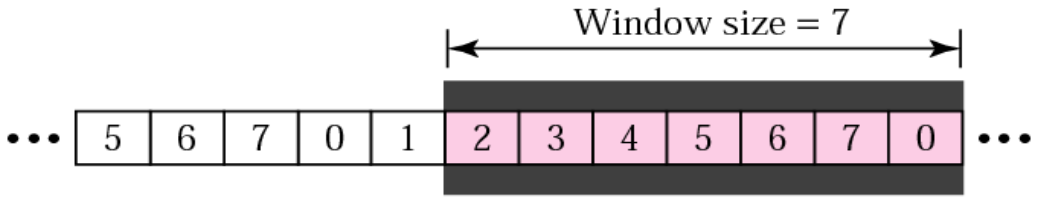


Working Principle

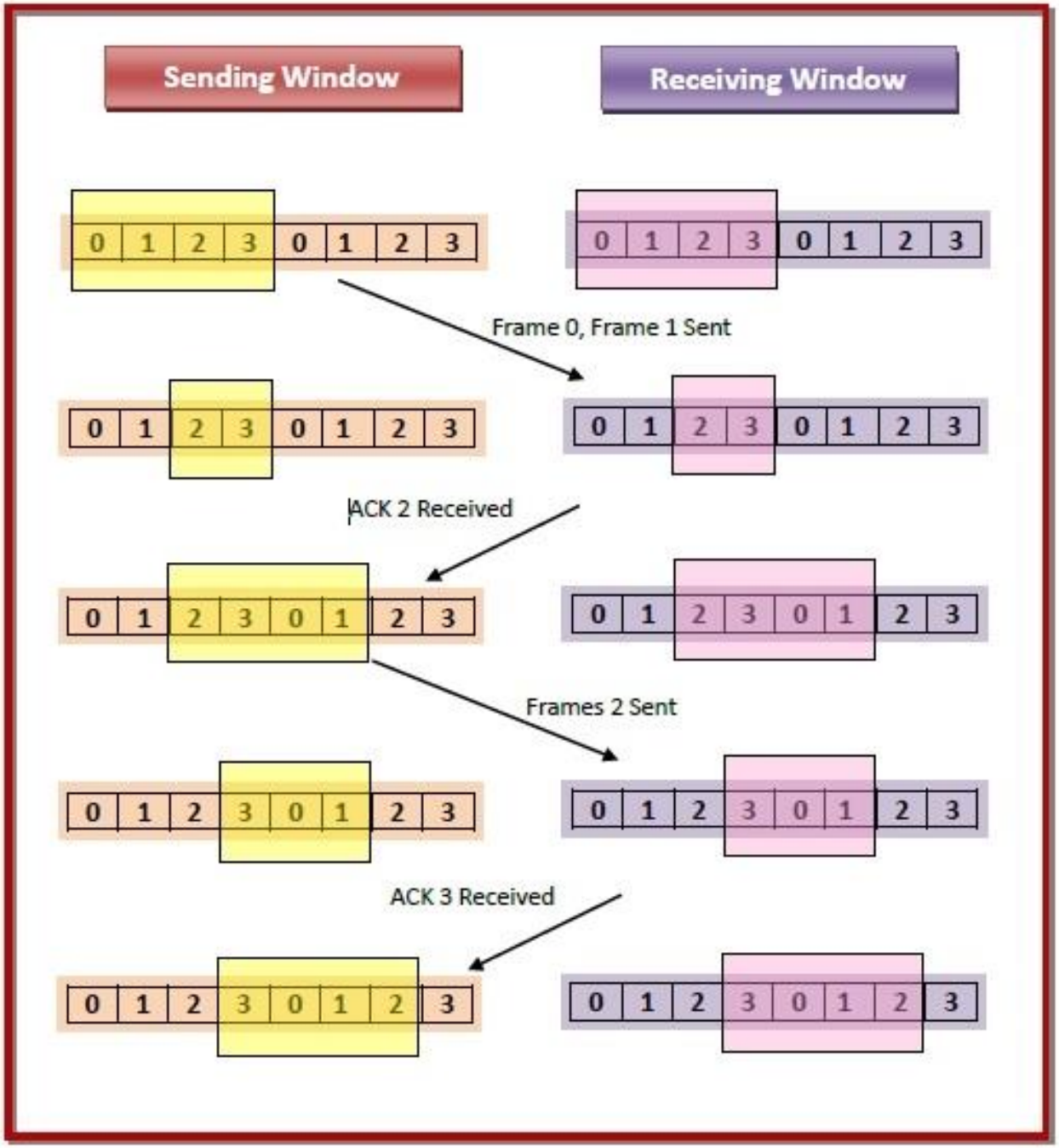
- In these protocols, the sender has a buffer called the sending window and the receiver has buffer called the receiving window.
- The size of the sending window determines the sequence number of the outbound frames.
- If the field is m bits long, the sequence numbers start from 0 to 2^m-1, and then are repeated.
- Therefore the window size is 2^m-1.



a. Before sliding



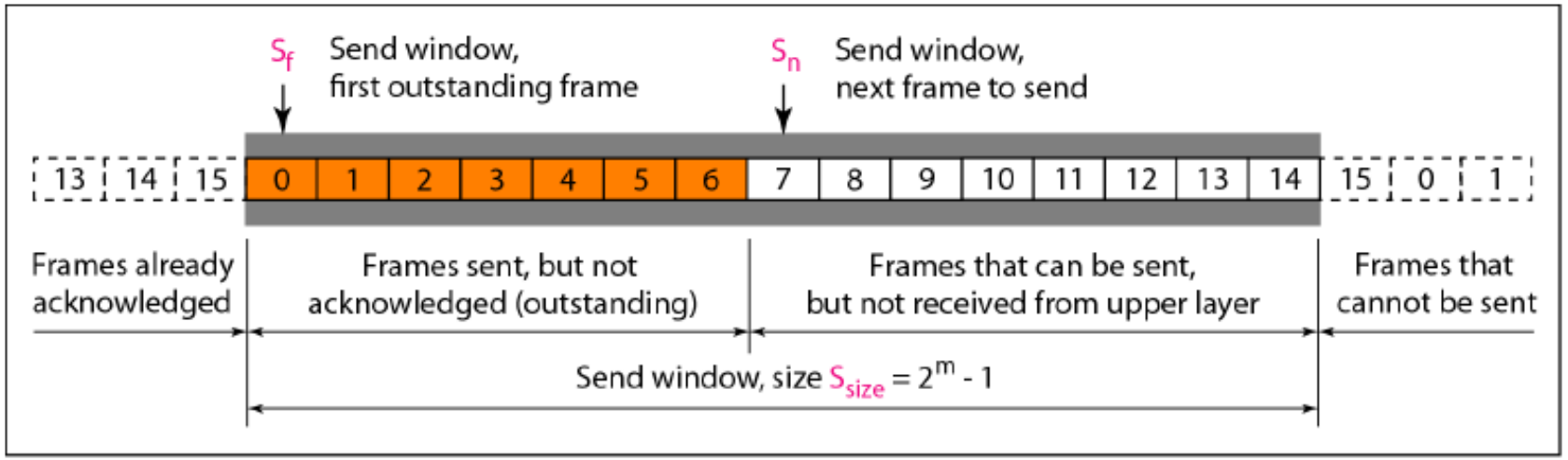
b. After sliding two frames


Sending Window
Receiving Window
0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3
Frame 0, Frame 1 Sent
0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3
ACK 2 Received
0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3
Frames 2 Sent
0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3
ACK 3 Received
0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3

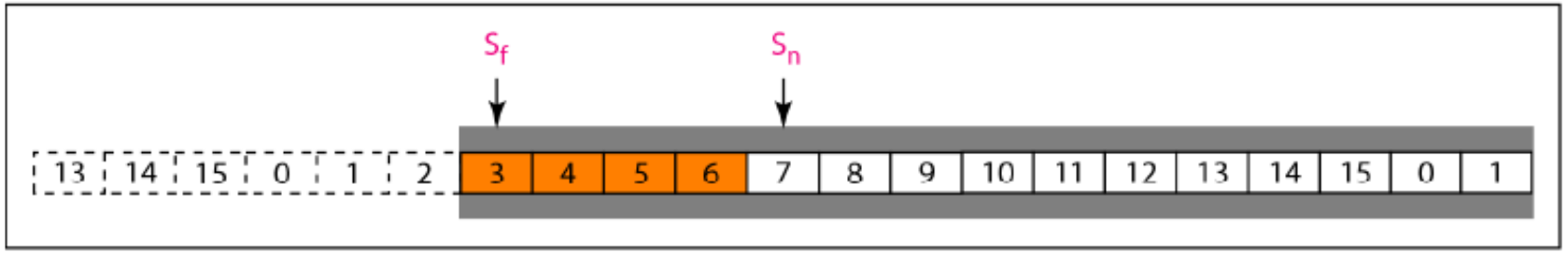
Go Back-*N* ARQ

- A station sends a series of frames sequentially up to a maximum number
- The number of unacknowledged frames outstanding is determined by window size, using the sliding window flow control technique.
- In case of no error, the destination will acknowledge incoming frames as usual.
- 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.

- 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.

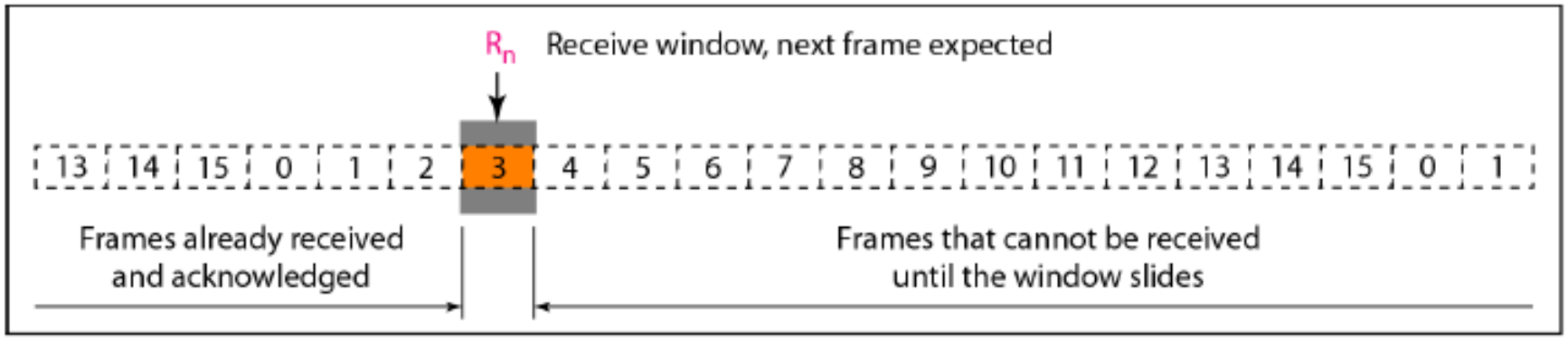


a. Send window before sliding

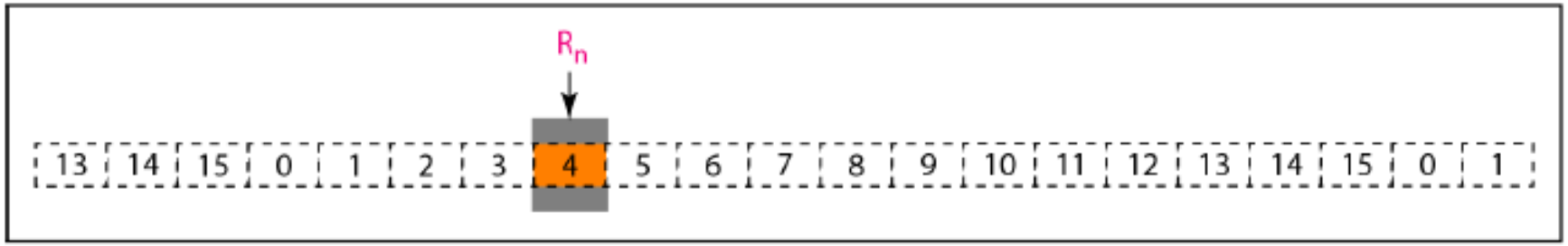


b. Send window after sliding

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.

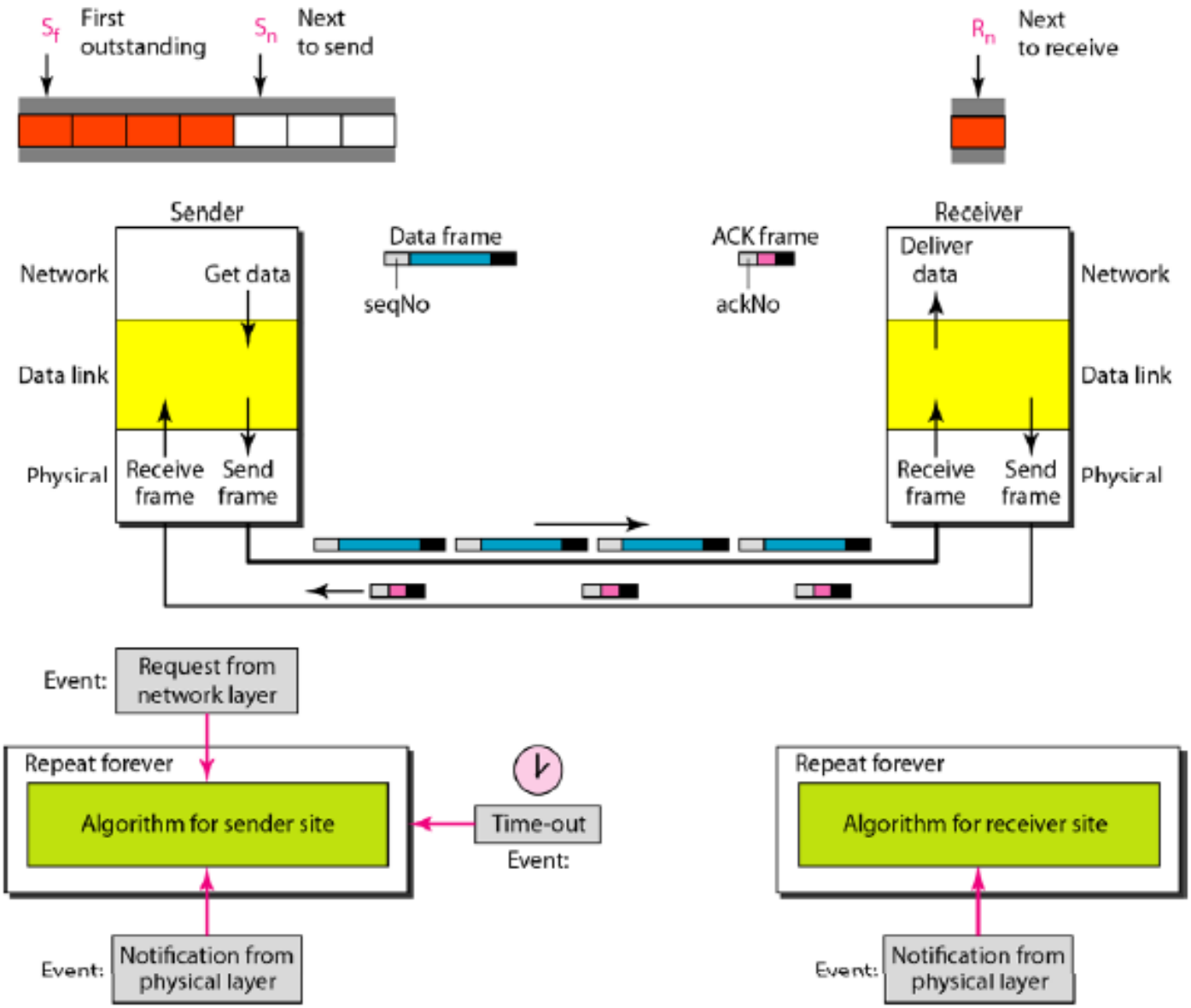


a. Receive window

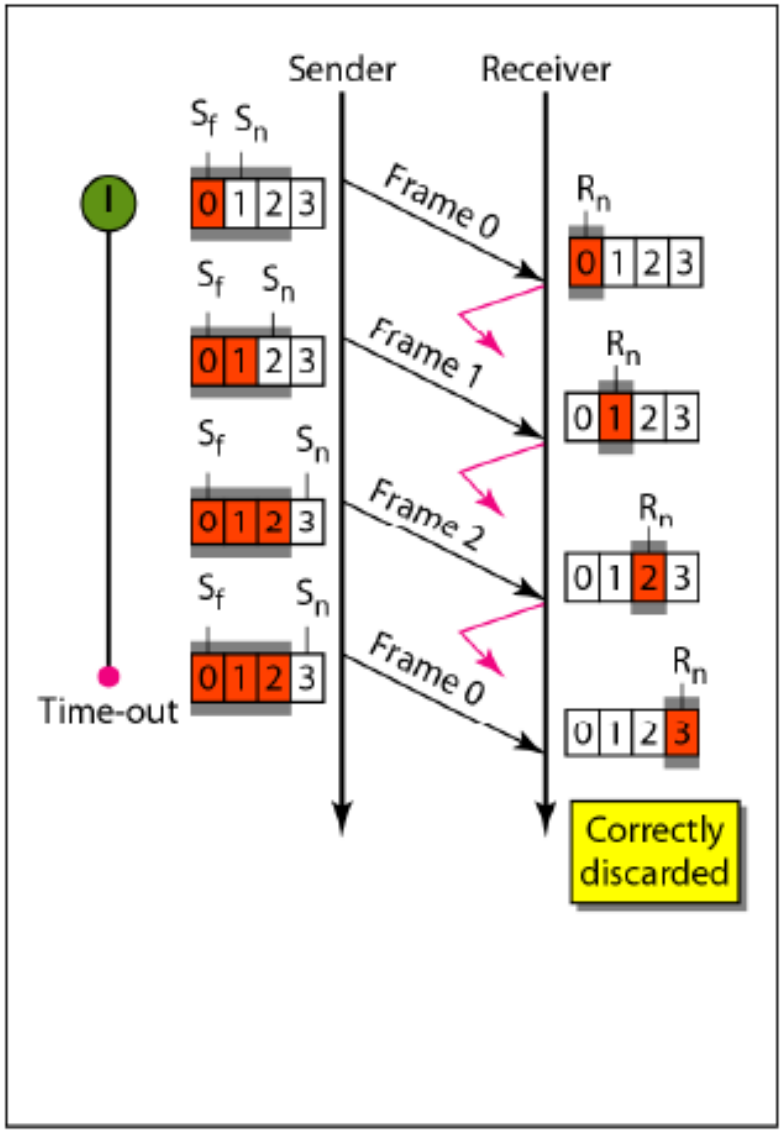


b. Window after sliding

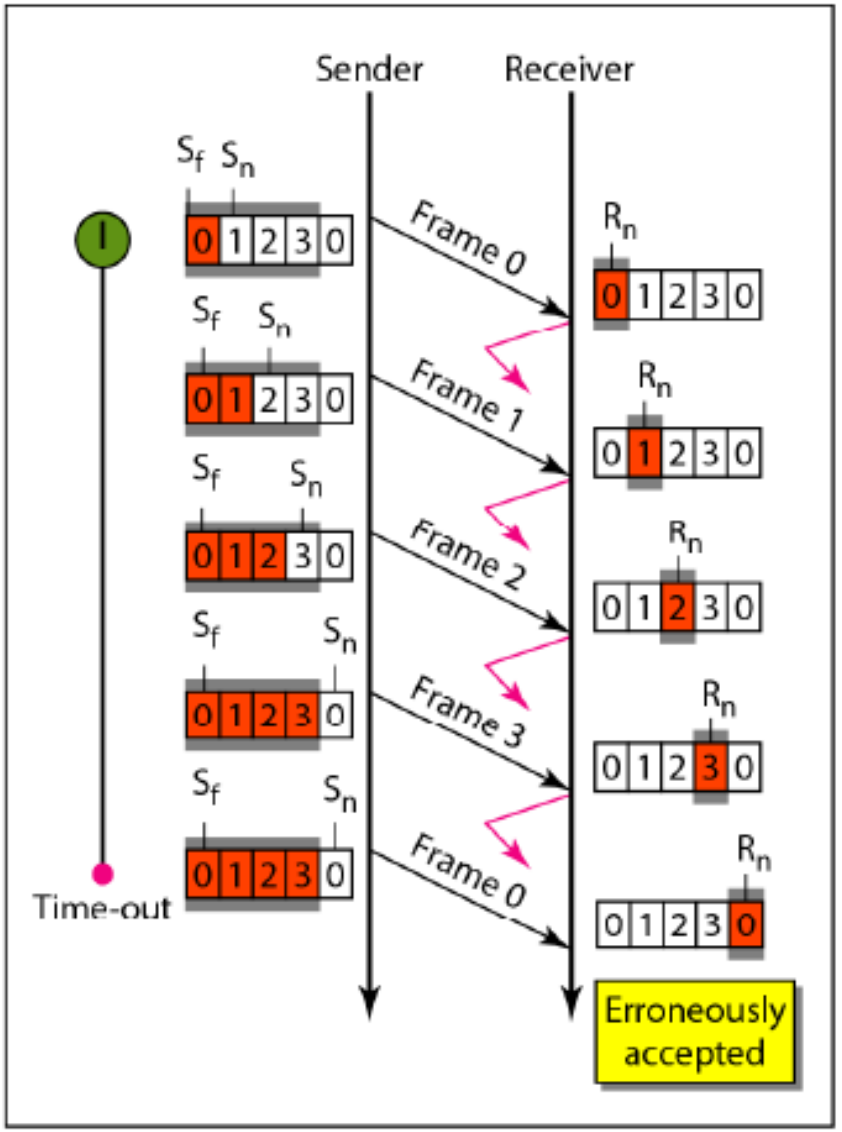
Design of Go-Back-N ARQ



Window size for Go-Back-N ARQ



a. Window size $< 2^m$



b. Window size $= 2^m$

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
```

(continued)

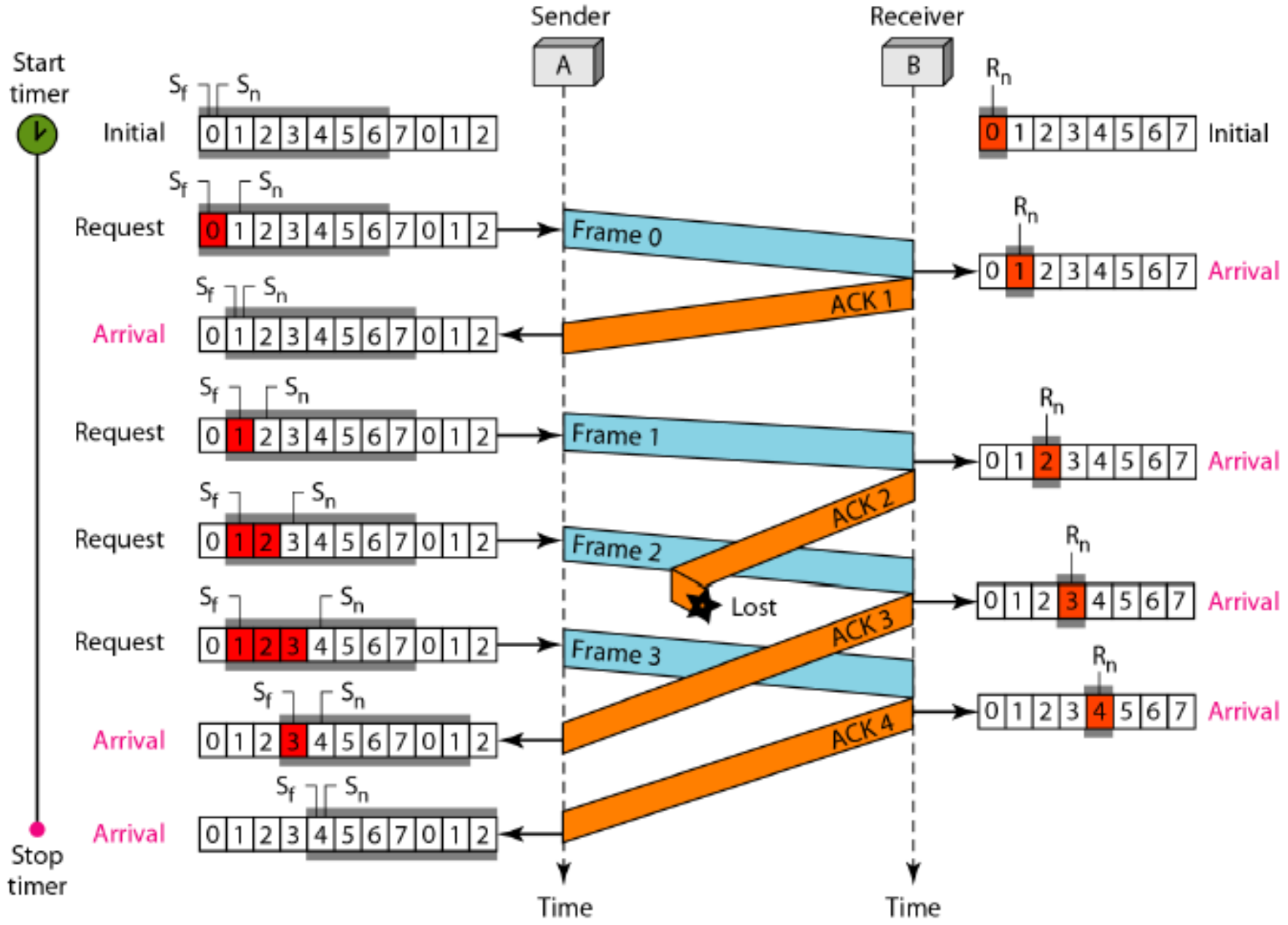
```
21    if(Event(ArrivalNotification)) //ACK arrives
22    {
23       Receive(ACK);
24       if(corrupted(ACK))
25            Sleep();
26       if((ackNo>Sf)&&(ackNo<=Sn)) //If a valid ACK
27       While(Sf <= ackNo)
28        {
29         PurgeFrame(Sf);
30         Sf = Sf + 1;
31        }
32        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 }
```

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 }
```

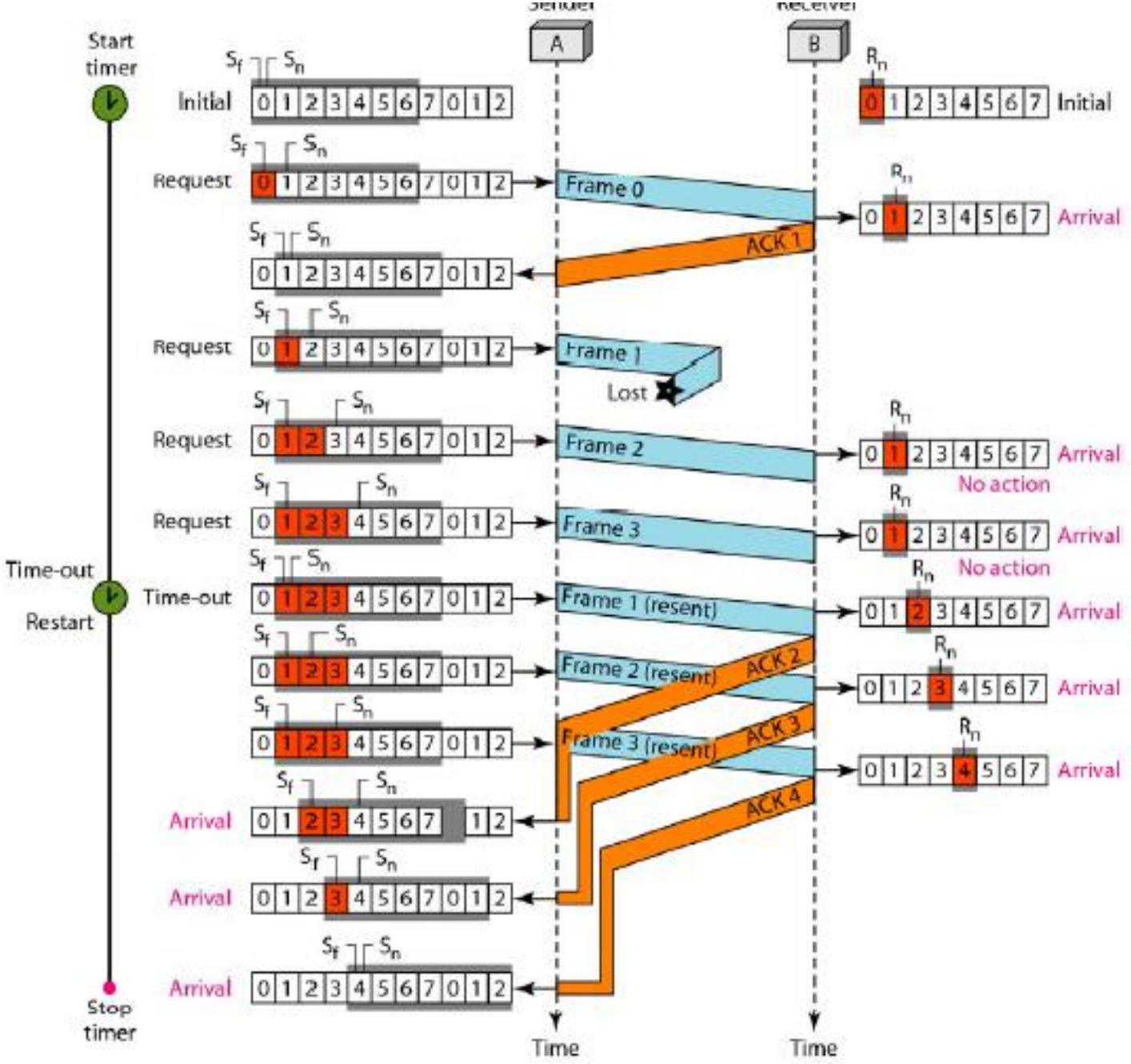
CASE-1

- *Figure on next slide shows an example of Go-Back-N ARQ flow control algorithm.*
- *This is an example of a case where the forward channel is reliable, but the reverse is not. No data frames are lost, but some ACKs are delayed and one is lost.*
- *The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.*
- *After initialization, there are seven sender events. Request events are triggered by data from the network layer; arrival events are triggered by acknowledgments from the physical layer.*
- *There is no time-out event here because all outstanding frames are acknowledged before the timer expires. Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.*


Sender
Receiver
A
B
Start timer
Stop timer
Initial
Request
Arrival
Frame 0
ACK 1
Frame 1
ACK 2
Frame 2
Lost
ACK 3
Frame 3
ACK 4
Time
Time

CASE-2

- *Figure on next slide shows what happens when a frame is lost Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost.*
- *The receiver receives frames 2 and 3, but they are discarded because they are received out of order.*
- *The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires.*
- *The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong. Note that the resending of frames 1, 2, and 3 is the response to one single event.*
- *When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3.*
- *The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state.*
- *A vertical line is used to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2.*
- *It happens again when ACK 4 arrives. Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.*

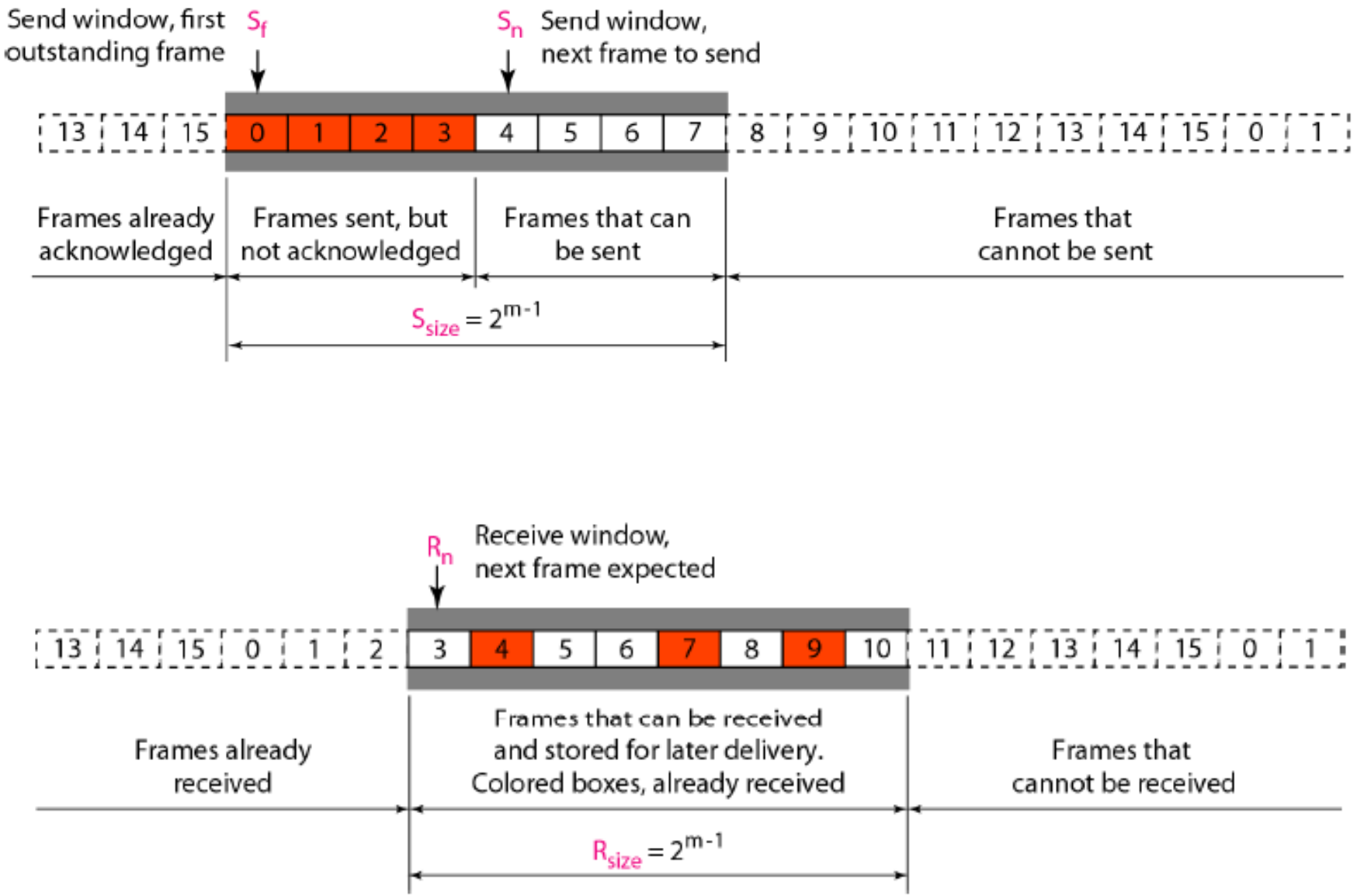

Sender
Receiver
A
B
Start timer
Initial
Request
Frame 0
ACK 1
Request
Frame 1
Lost
Request
Frame 2
Request
Frame 3
Time-out
Restart
Time-out
Frame 1 (resent)
Frame 2 (resent)
ACK 2
Frame 3 (resent)
ACK 3
ACK 4
Arrival
Arrival
Arrival
Stop timer
Time
Time
Initial
Arrival
Arrival
No action
Arrival
No action
Arrival
Arrival
Arrival

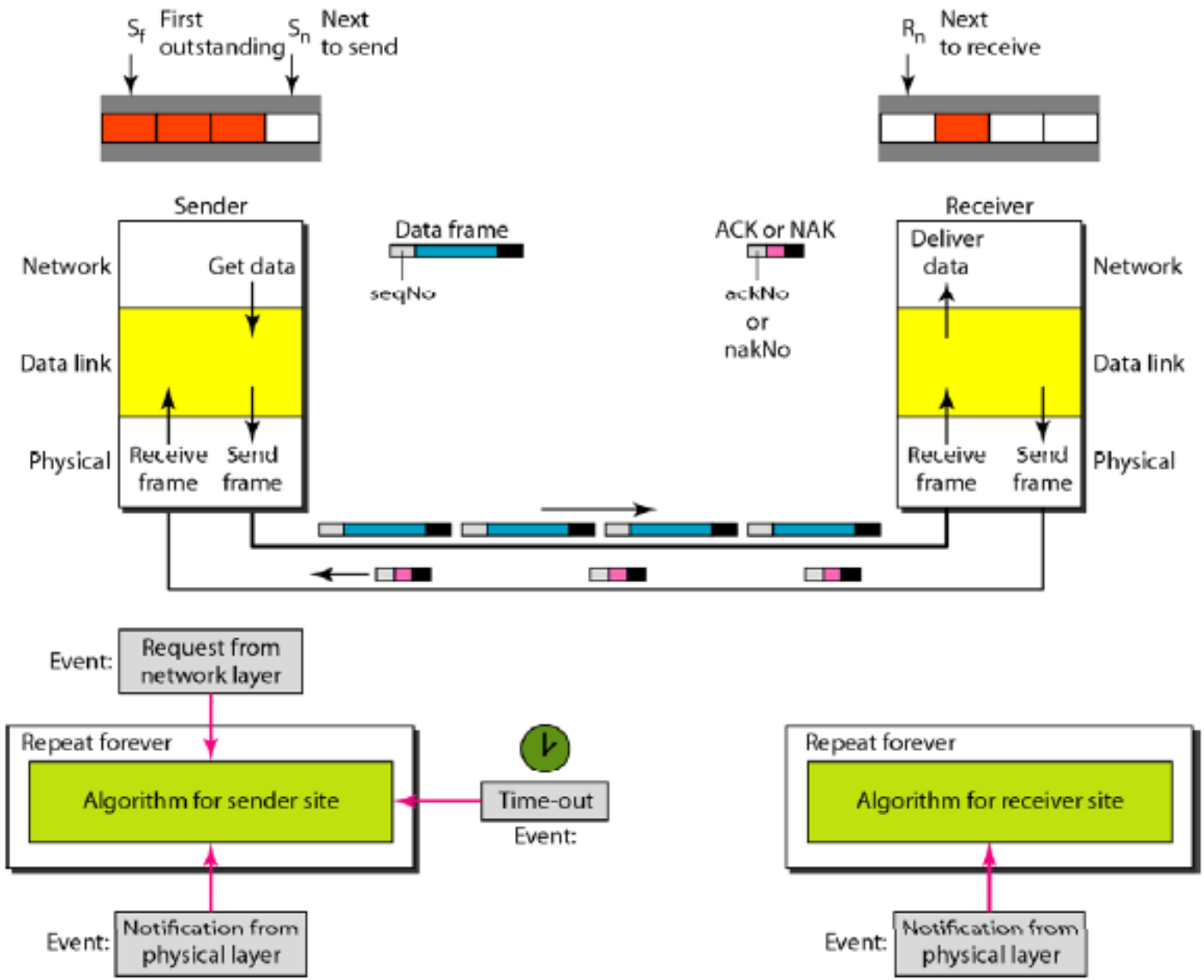
Performance of Go-Back-N ARQ

- *Go-Back-N* ARQ simplifies the process at the receiver site.
- The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded.
- However, this protocol is very inefficient for a noisy link. 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.

Selective Repeat ARQ

- 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 Repeat ARQ.
- It is more efficient for noisy links, but the processing at the receiver is more complex.
- The Selective Repeat Protocol also uses two windows: a send window and a receive window.
- **In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m-1.**


Send window, first outstanding frame
S_f
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged
Frames that can be sent
Frames that cannot be sent
$S_{size} = 2^{m-1}$
R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received
Frames that can be received and stored for later delivery. Colored boxes, already received
Frames that cannot be received
$R_{size} = 2^{m-1}$


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 or NAK
ackNo
or
nakNo
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

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
```

(continued)

```
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 }
```

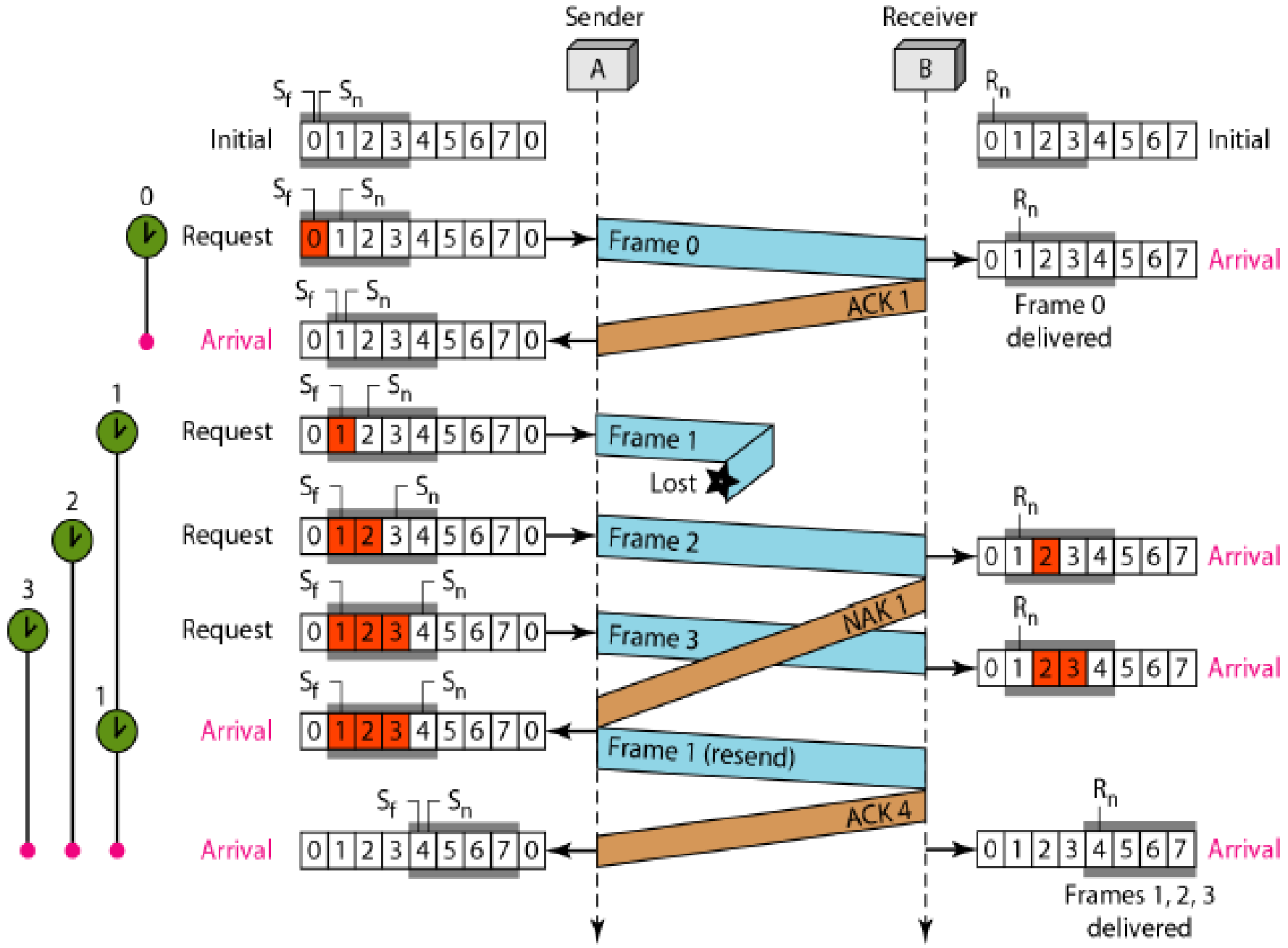
Receiver-site Selective Repeat algorithm

```
 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(R_n))
29           {
30            DeliverData(R_n);
31            Purge(R_n);
32            R_n = R_n + 1;
33            AckNeeded = true;
34           }
35            if(AckNeeded);
36            {
37            SendAck(R_n);
38            AckNeeded = false;
39            NakSent = false;
40            }
41          }
42         }
43     }
44 }
```

- *Figure on the next slides shows the situation. One main difference is the number of timers.*
- *Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1, 2, and 3). The timer for frame 0 starts at the first request, but stops when the ACK for this frame arrives.*
- *The timer for frame 1 starts at the second request, restarts when a NAK arrives, and finally stops when the last ACK arrives.*
- *The other two timers start when the corresponding frames are sent and stop at the last arrival event.*
- *At the receiver site we need to distinguish between the acceptance of a frame and its delivery to the network layer.*
- *At the second arrival, frame 2 arrives and is stored and marked, but it cannot be delivered because frame 1 is missing.*
- *At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered.*
- *Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer.*
- *There are two conditions for the delivery of frames to the network layer:*
  - *First, a set of consecutive frames must have arrived.*
  - *Second, the set starts from the beginning of the window.*

- *Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same.*
- *The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames.*
- *The second NAK would still be NAK1 to inform the sender to resend frame 1 again; this has already been done.*
- *The first NAK sent is remembered (using the nakSent variable) and is not sent again until the frame slides. A NAK is sent once for each window position and defines the first slot in the window.*
- *The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames.*
- *In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to n frames are delivered in one shot, only one ACK is sent for all of them.*

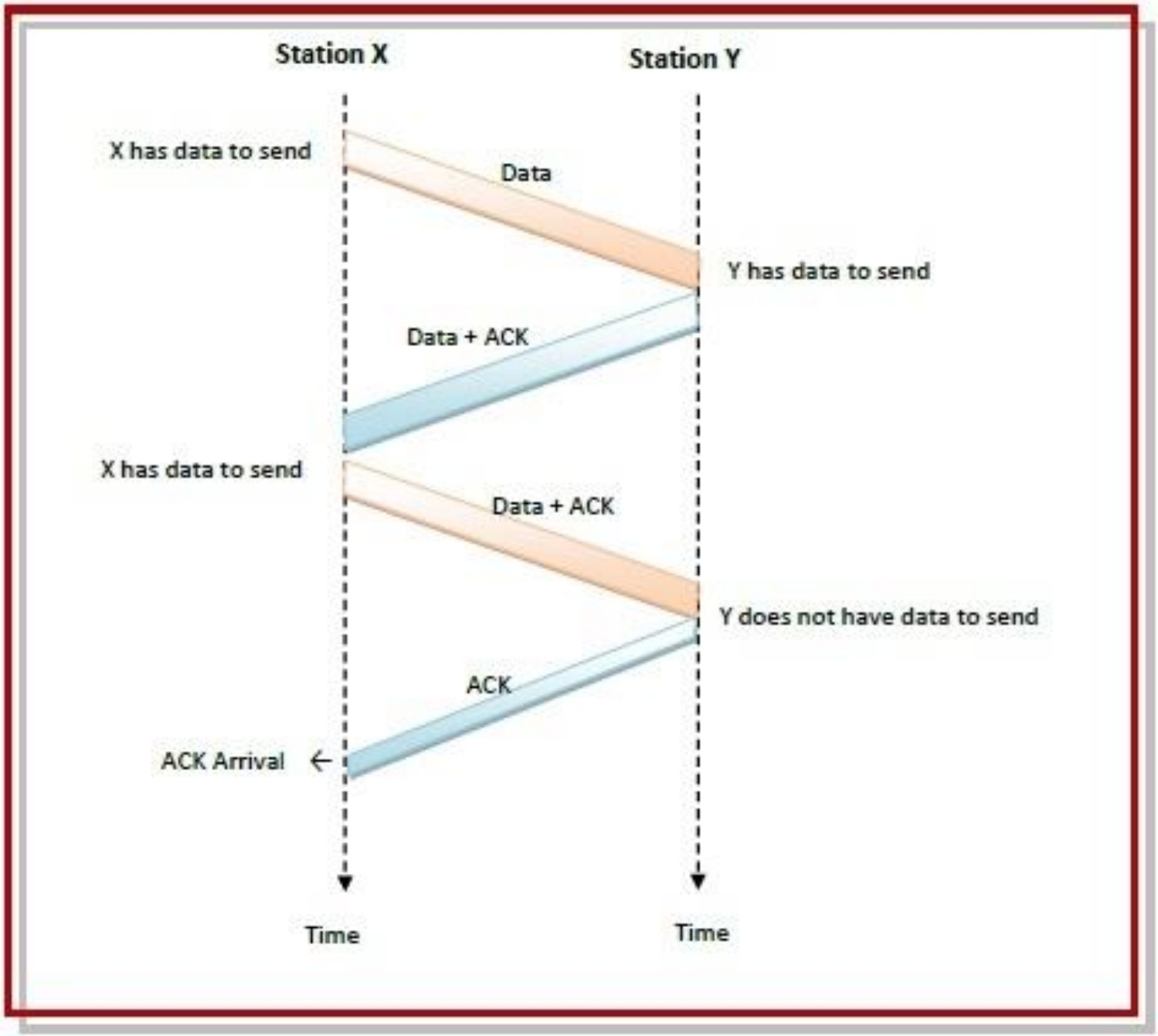

Sender
Receiver
A
B
Initial
Request
Arrival
Frame 0
ACK 1
Frame 0 delivered
Frame 1
Lost
Frame 2
Frame 3
NAK 1
Frame 1 (resend)
ACK 4
Frames 1, 2, 3 delivered

Piggybacking

- In reliable full - duplex data transmission, the technique of hooking up acknowledgments onto outgoing data frames is called piggybacking.
- Communications are mostly full – duplex in nature, i.e. data transmission occurs in both directions.
- A method to achieve full – duplex communication is to consider both the communication as a pair of simplex communication.
- Each link comprises a forward channel for sending data and a reverse channel for sending acknowledgments.
- However, in the above arrangement, traffic load doubles for each data unit that is transmitted. Half of all data transmission comprise of transmission of acknowledgments.
- Piggybacking a solution that provides better utilization of bandwidth.

Working Principle

- Suppose that there are two communication stations X and Y. The data frames transmitted have an acknowledgment field, *ack* field that is of a few bits length. Additionally, there are frames for sending acknowledgments, ACK frames. The purpose is to minimize the ACK frames.
- The three principles governing piggybacking when the station X wants to communicate with station Y are:
  - If station X has both data and acknowledgment to send, it sends a data frame with the *ack* field containing the sequence number of the frame to be acknowledged.
  - If station X has only an acknowledgment to send, it waits for a finite period of time to see whether a data frame is available to be sent. If a data frame becomes available, then it piggybacks the acknowledgment with it. Otherwise, it sends an ACK frame.
  - If station X has only a data frame to send, it adds the last acknowledgment with it. The station Y discards all duplicate acknowledgments. Alternatively, station X may send the data frame with the *ack* field containing a bit combination denoting no acknowledgment.


Station X
Station Y
X has data to send
Data
Y has data to send
Data + ACK
X has data to send
Data + ACK
Y does not have data to send
ACK
ACK Arrival
Time
Time

Design of piggybacking in Go-Back-N ARQ