Reliable Data Transfer (RDT)

Manas Ranjan Lenka
School of Computer Engineering,
KIIT University

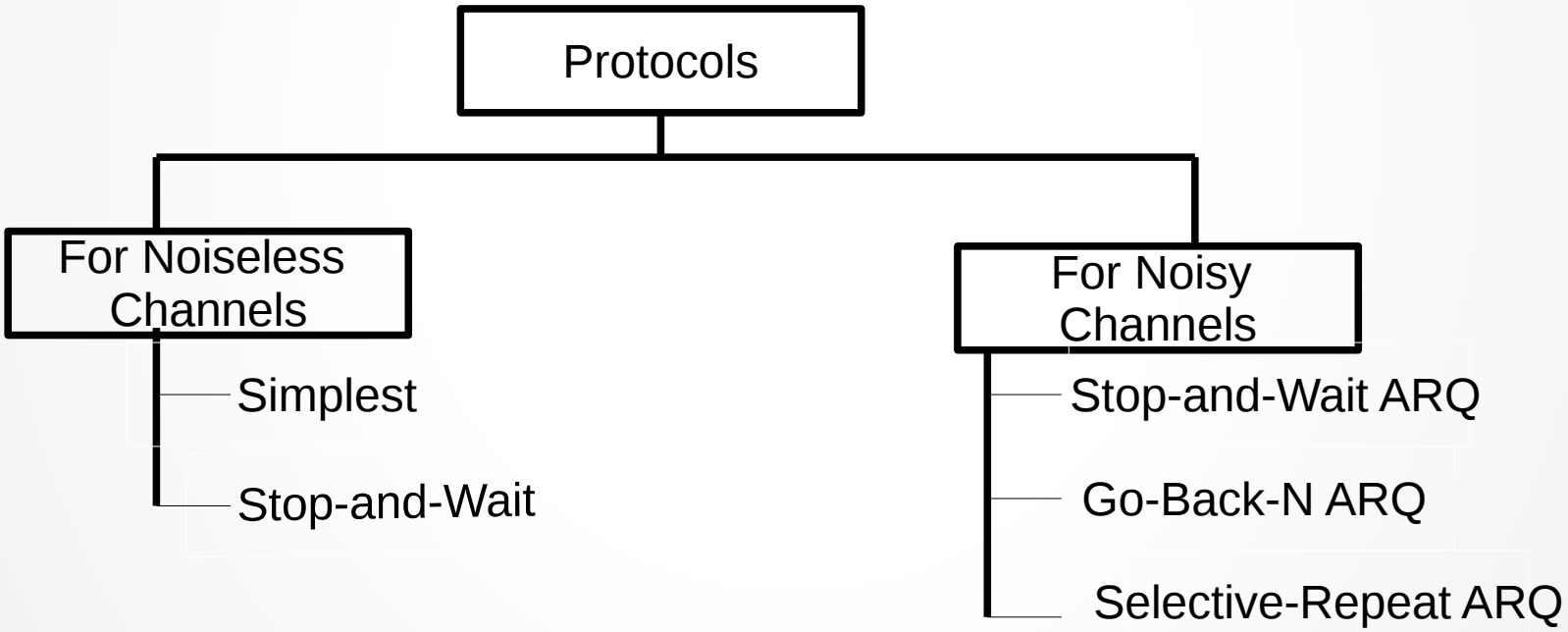
Flow Control & Error Control

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

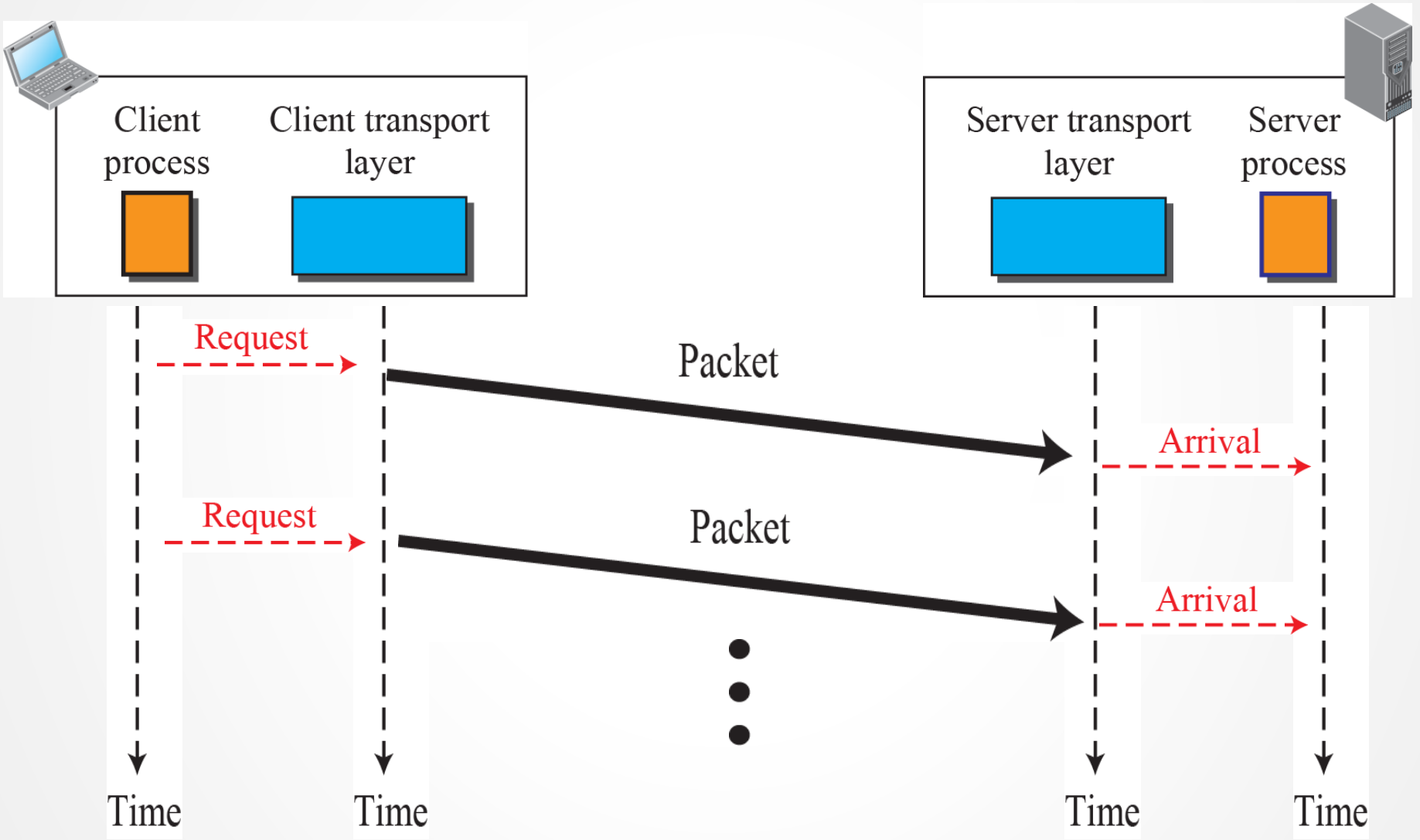
- **Error control** is based on automatic repeat request, which is the retransmission of data.

Protocols

- Transport layer can combine flow control, and error control to achieve the delivery of data from one node to another.



Simplest Protocol



Simplest Protocol

Sender-site algorithm for the simplest protocol

```
while (true)       //Repeat forever
{
    WaitForEvent();                 //Sleep until an event occurs
    if(Event(RequestToSend))    //There is a packet to send
    {
        GetData();
        MakeSegment();
        SendSegment();          //Send the Segment
    }
}
```

Simplest Protocol

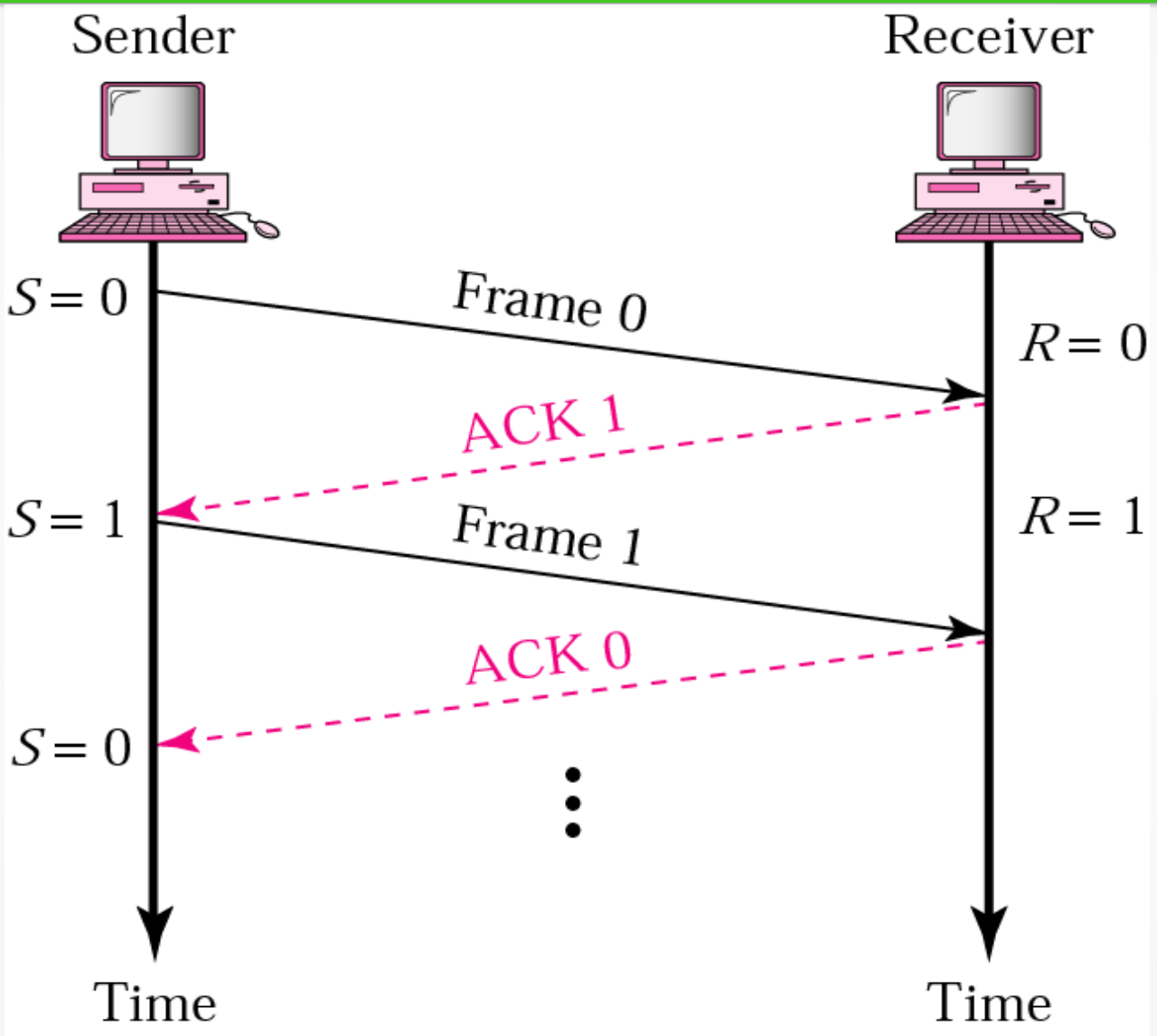
Receiver-site algorithm for the simplest protocol

```
while (true)       //Repeat forever
{
    WaitForEvent();                 //Sleep until an event occurs
    if(Event(ArrivalNotification))       //Data Segment arrived
    {
        ReceiveSegment();
        ExtractData();
        DeliverData ( );          //Deliver data to application layer
    }
}
```

Problem with Simplest Protocol

- If data segment arrive at the receiver site faster than they can be processed.
- This may result in either the discarding of segments or denial of service.
- To prevent the receiver from becoming over-whelmed with segments,we somehow need to tell the sender to slow down.
- There must be feedback from the receiver to the sender.

Stop-and-Wait Protocol



Stop-and-Wait Protocol

- We add flow control to our previous protocol.
- sender sends one segment, stops until it receives confirmation from the receiver and then sends the next segment.

Stop-and-Wait Protocol

Sender-site algorithm

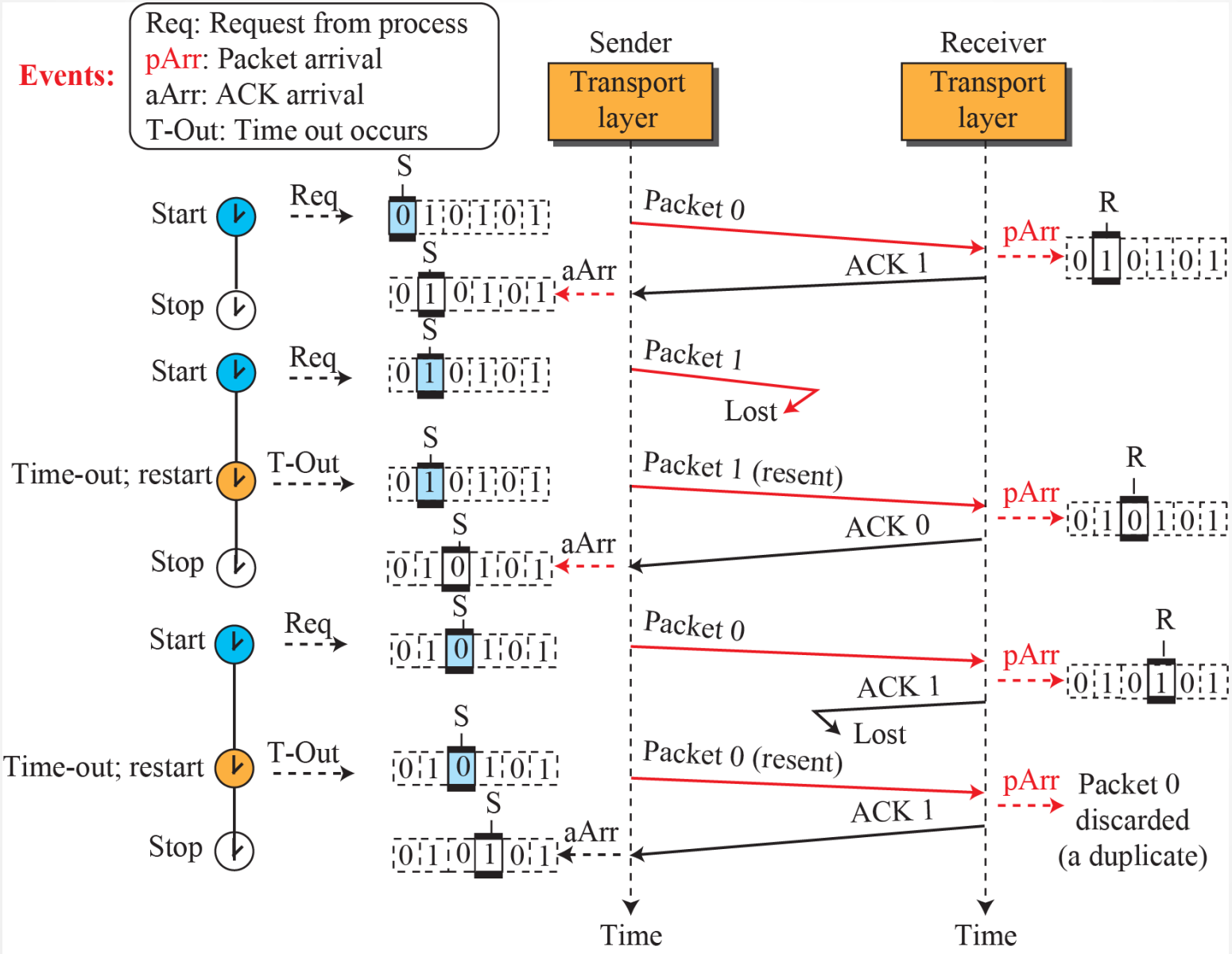
```
canSend = true
while (true)            //Repeat forever
{
    WaitForEvent();             //Sleep until an event occurs
    if(Event(RequestToSend) && canSend)  //There is a packet to send
    {
        GetData();
        MakeSegment();
        SendSegment();      //Send the Segment
        canSend = false;    //cannot send until ACK arrives
    }
    WaitForEvent();             //Sleep until an event occurs
    if(Event(ArrivalNotification))  //An ACK has received
    {
        ReceiveSegment();       //Receive the ACK frame
        canSend = true;
        }
    }
```

Stop-and-Wait Protocol

Receiver-site algorithm

```
while (true)      //Repeat forever
{
    WaitForEvent();         //Sleep until an event occurs
    if(Event(ArrivalNotification))       //Data Segment arrived
    {
        ReceiveSegment();
        ExtractData();
        DeliverData ( );         //Deliver data to application layer
        SendSegment();     //Send an Ack Segment
    }
}
```

Stop-and-Wait ARQ



Stop-and-Wait ARQ

- adds a simple error control mechanism to the Stop-and-Wait Protocol
- error control deals with 3 types of segments
  - Corrupted segments (Silence)
  - Lost segments (Retransmit)
  - Duplicate segments (Sequence No.)

Sequence Numbers

- Range of Sequence no. with m bits (0 to 2^m-1).
- Sequence Number Range for Stop-And-Wait ARQ
  - The Segment x arrives safe at the receiver and receiver sends an acknowledgment which arrives safe at the sender. The sender to send the next frame numbered x + 1.
  - The Segment is corrupted or never arrives at the receiver ; the sender resends the frame (numbered x) after the time-out.
  - The Segment x arrives safe at the receiver site; the receiver sends an acknowledgment, which is corrupted or lost. The sender resends the frame (numbered x) after the time-out. Note that the frame here is a duplicate. The receiver can recognize this fact because it expects frame x + 1 but frame x was received.

Stop-and-Wait ARQ

Sender-site algorithm

```
Sn=0                    // Frame 0 should be sent first
canSend = true
 while (true)           //Repeat forever
 {
    WaitForEvent();     //Sleep until an event occurs
    if(Event(RequestToSend) && canSend)  //There is a packet to send
    {
            GetData();
            MakeSegment(Sn);
            StoreSegment(Sn);    //Keep the copy
            SendSegment(Sn);     //Send the Segment
            StartTimer();
            Sn=Sn+1;
            canSend = false;     //cannot send until ACK arrives
    }
```

Stop-and-Wait ARQ

Sender-site algorithm ...

```
WaitForEvent();                 //Sleep until an event occurs
if(Event(ArrivalNotification))  //An ACK has received
{
        ReceiveSegment(ackNo);          //Receive the ACK frame
        if(not corrupted AND ackNo == Sn)   // valid ACK
        {
            StopTimer();
            DeleteSegment(Sn-1);        //copy no more needed
            canSend=true;
        }
}
if (Event (TimeOUt)                             //The timer expired
{
            StartTimer();
            ResendSegment(Sn_l);    //Resend a copy check
}
}
```

Stop-and-Wait ARQ

Receiver-site algorithm

```
Rn=0;
while (true)      //Repeat forever
{
    WaitForEvent();         //Sleep until an event occurs
    if(Event(ArrivalNotification))       //Data Segment arrived
    {
        ReceiveSegment();
        if (!corruptedSegment() )
        {
            if(seqNo == Rn)
 –          {
            ExtractData();
            DeliverData ( );         //Deliver data to application layer
            Rn=Rn+1;
 –          }
            SendACKSegment(Rn);  //Send an Ack Segment
        }
    }
}
```

Efficiency of Stop-And-Wait Protocol

1. Transmission time: The time it takes for a station to transmit a frame (normalized to a value of 1).
2. Propagation delay: The time it takes for a bit to travel from sender to receiver.

a = Propagation time / transmission time

– a < 1 : The frame is sufficiently long such that the first bits of the frame arrive at the destination before the source has completed transmission of the frame.

– a > 1: Sender completes transmission of the entire frame before the leading bits of the frame arrive at the receiver.

The link utilization U = 1/(1+2a)

When the propagation time is small, as in case of LAN environment, the link utilization is good.

But, in case of long propagation delays, as in case of satellite communication, the utilization can be very poor.

To improve the link utilization, we can use the following (sliding-window) protocol instead of using stop-and-wait protocol.

Efficiency of Stop-And-Wait Protocol

1. Assume that, in a Stop-and-Wait system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 milliseconds to make a round trip. What is the bandwidth-delay product? If the system data packets are 1,000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is $(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000$ bits.

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and the acknowledgment to come back.

However, the system sends only 1,000 bits.

The link utilization is only 1,000/20,000, or 5 percent.

N.B. - For a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.

Efficiency of Stop-And-Wait Protocol

1. The distance from earth to a distant planet is approximately 9 × 10^10 m. What is the channel utilization if a stop-and-wait protocol is used for frame transmission on a 64 Mbps point-to-point link? Assume that the frame size is 32 KB and the speed of light is 3 × 10^8 m/s.

Solution

Propagation delay = (9 × 10^10 m) / (3 × 10^8 m/s) = 300s

RTT = 2*Propagation delay = 600s

No. of bits within 1 RTT = (64 x 10^6) x 600 = 384 x 10^8

Utilization = (32 x 10^3 x 8) / (384 x 10^8) = 1.5 x 10^-5

Efficiency of Stop-And-Wait Protocol

1. *In the previous problem, suppose a sliding window protocol is used instead. For what sender window size will the link utilization be 100%? You may ignore the protocol processing times at the sender and the receiver.*

Solution

we can send 1.5×10^5 packets in 1 RTT
So window size should be 150000

Efficiency of Stop-And-Wait ARQ

Consider the use of 10 K-bit size frames on a 10 Mbps satellite channel with 270 ms delay. What is the link utilization for stop-and-wait ARQ technique assuming $P = 10^{-3}$?

Solution:

Link utilization = (1-P) / (1+2a)

Where a = (Propagation Time) / (Transmission Time)

Propagation time = 270 msec

Transmission time = (frame length) / (data rate)

= (10 K-bit) / (10 Mbps)

= 1 msec

Hence, a = 270/1 = 270

Link utilization = 0.999/(1+2*270) ≈0.0018 =0.18%

Problem with Stop-and-Wait ARQ

- No pipelining :
  - waits for a frame to reach the destination and be acknowledged before the next frame can be sent.
- To improves the efficiency of the transmission pipelining needs to be supported.
- Using the following protocol we achieve pipelining.
  - Go-Back-N Automatic Repeat Request
  - Selective Repeat Automatic Repeat Request

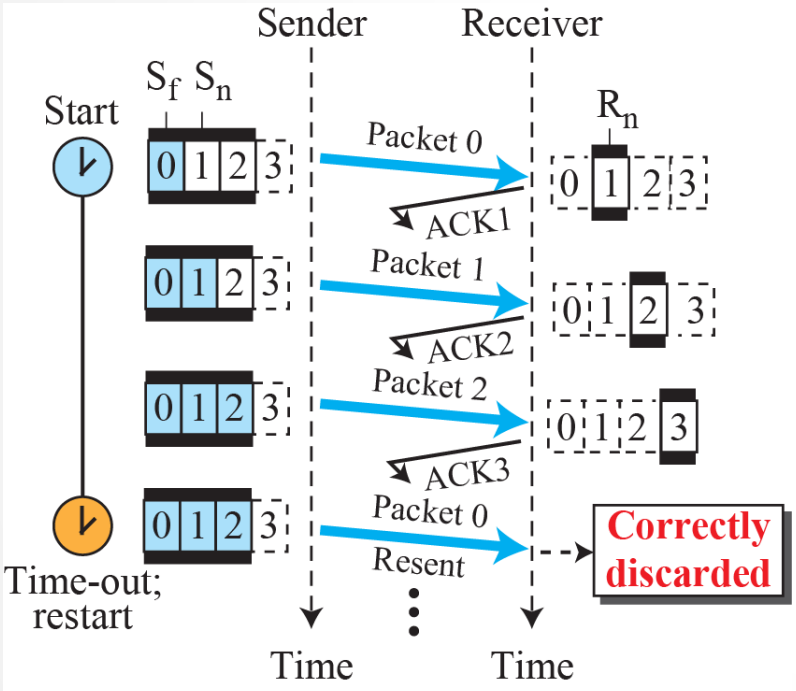
Go-Back-N Automatic Repeat Request

- To improve the efficiency of transmission, multiple packets must be in transit while the sender is waiting for acknowledgment.
- In this protocol we can send several frames before receiving acknowledgments;
- we keep a copy of these frames until the acknowledgments arrive.

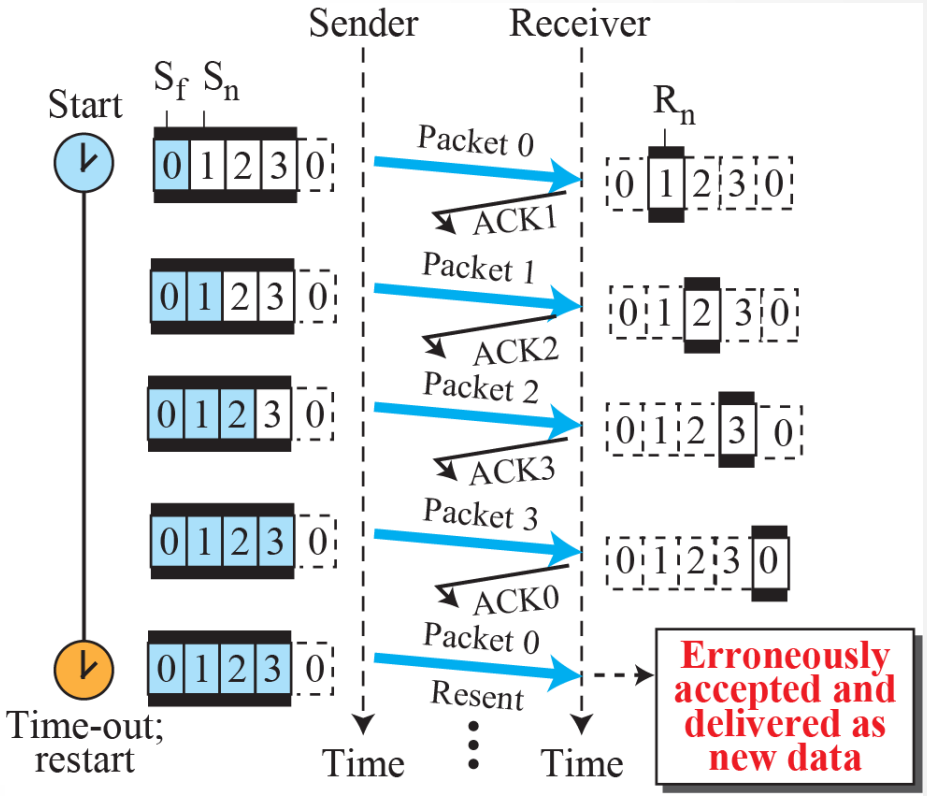
Go-Back-N Automatic Repeat Request

- **Sequence Numbers:** Range of Sequence no. with m bits (0 to 2^m-1). In other words, the sequence numbers are modulo-2^m .
- **Sliding Window:** the range of sequence numbers that is the concern of the sender and receiver.
  - The range which is the concern of the sender is called the send sliding window. The size of the send window is $2^m - 1$.
  - The range that is the concern of the receiver is called the receive sliding window. The size of the receive window is 1.

Send window size for Go-Back-N Automatic Repeat Request

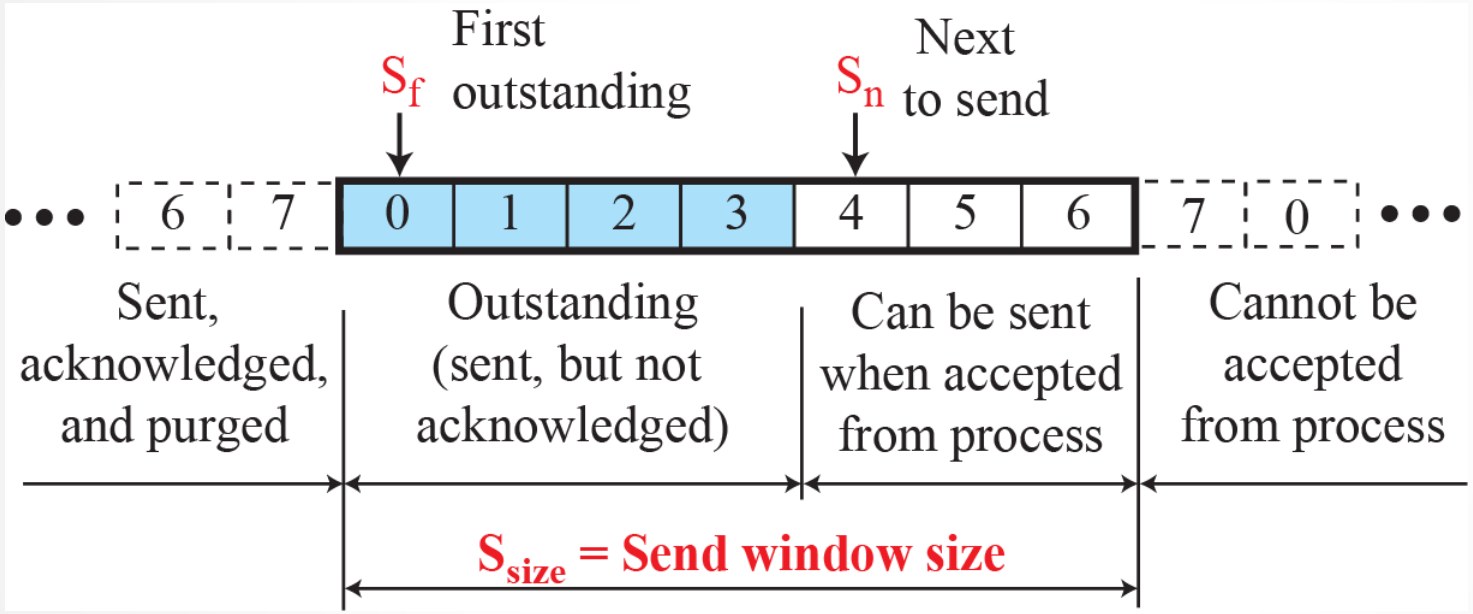


a. Send window of size < 2^m

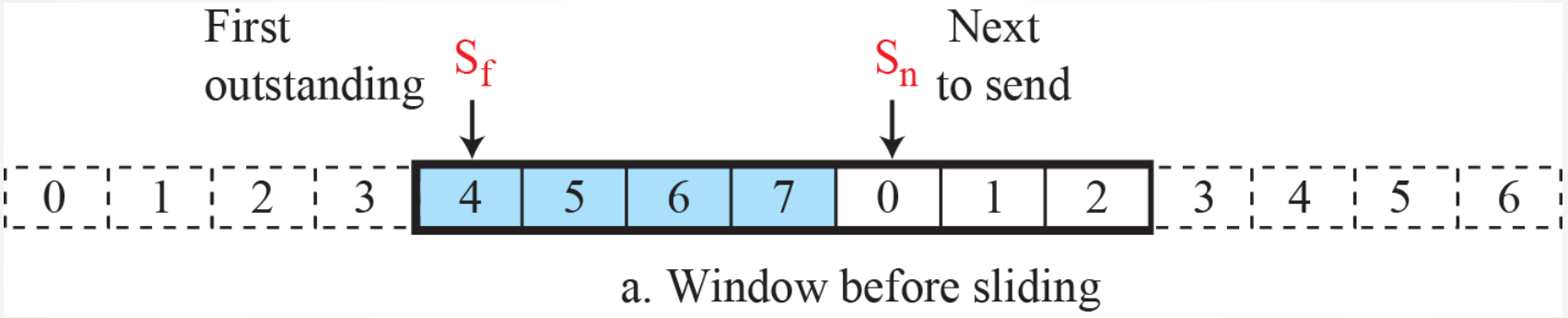


b. Send window of size = 2^m

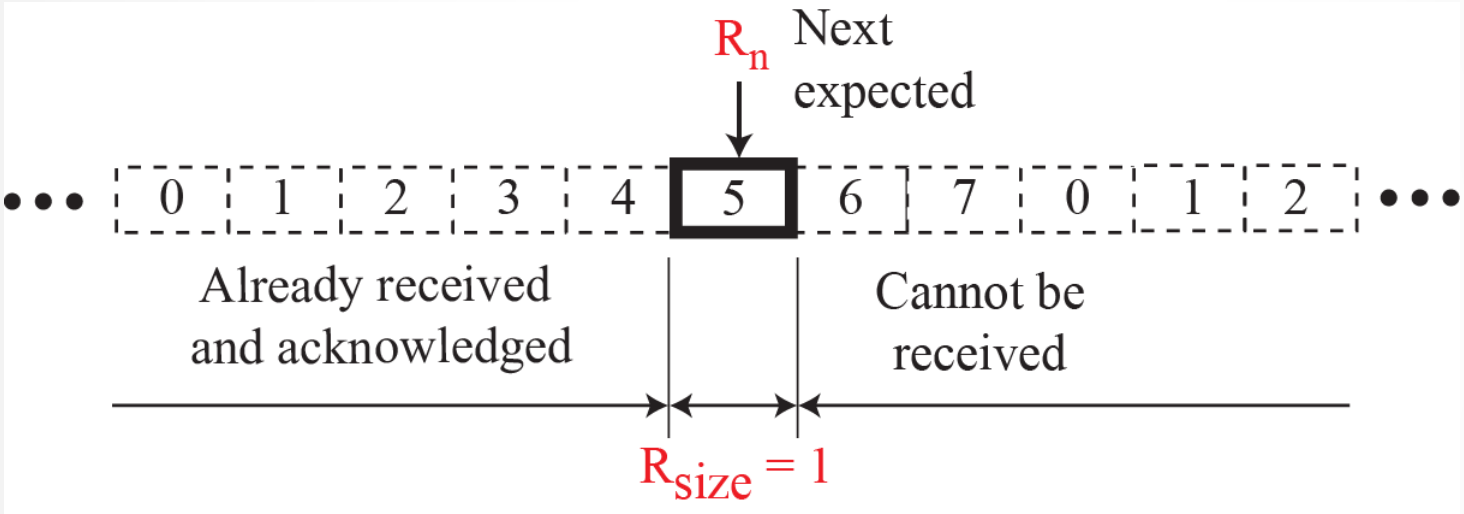
Send window for Go-Back-N Automatic Repeat Request



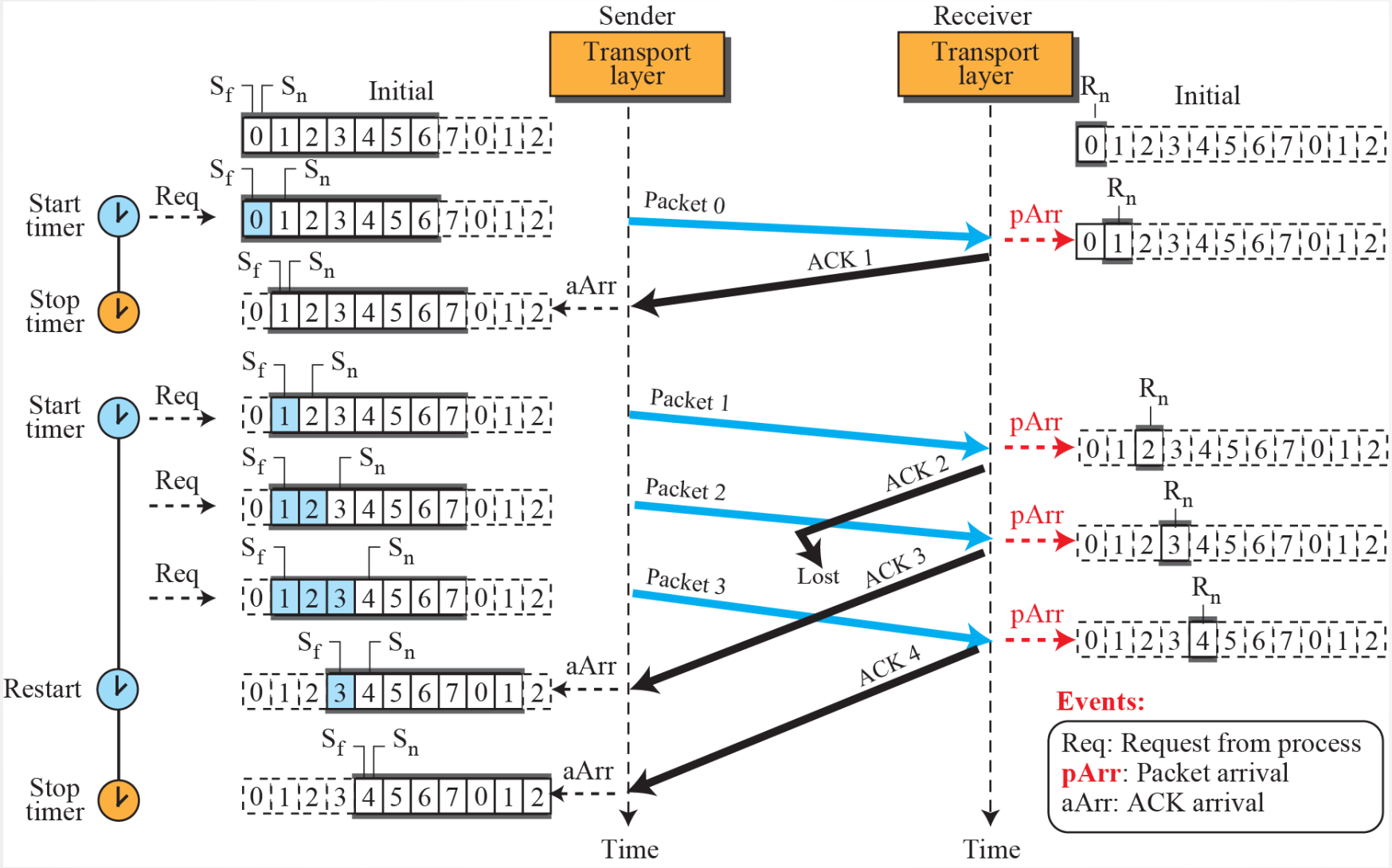
Sliding the send window



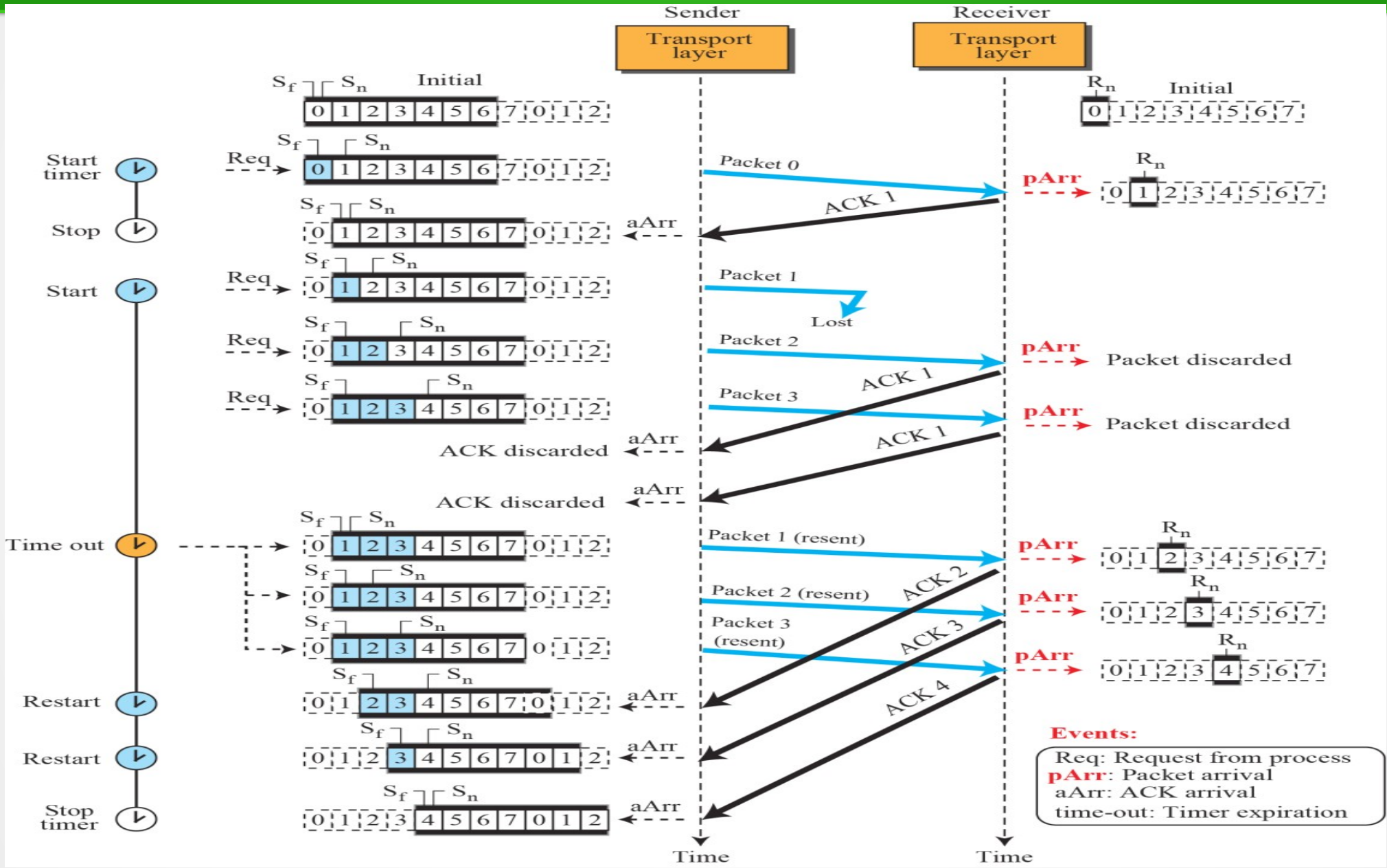
Receive window for Go-Back-N



Example of Go-Back-N ARQ



Example of Go-Back-N ARQ



Go-Back-N ARQ

Sender-site algorithm

S_w = 2''' - 1;

```
Sn=0;
Sf = 0;                    // Frame 0 should be sent first
while (true)               //Repeat forever
    {
        WaitForEvent();        //Sleep until an event occurs
        if(Event(RequestToSend) )  //There is a packet to send
        {
                if (Sn-Sf < Sw) //If window is not full
                {
                    GetData();
                    MakeSegment(Sn);
                    StoreSegment(Sn);     //Keep the copy
                    SendSegment(Sn);      //Send the Segment
                    Sn=Sn+1;
                    if (timer not running)
                        StartTimer();
                }
        }
```

Go-Back-N ARQ

Sender-site algorithm ...

```
WaitForEvent();    //Sleep until an event occurs
    if(Event(ArrivalNotification))  //An ACK has received
    {
        ReceiveSegment(ackNo);    //Receive the ACK frame
        if(not corrupted ACK && ackNo > Sf && ackNo <= Sn)   // valid ACK
        {
            While(Sf <= ackNo)
            {
                DeleteSegment(Sf);    //copy no more needed
                Sf = Sf + 1;
            }
            StopTimer();
        }
    }
    if (Event (TimeOUt)       //The timer expired
    {
            StartTimer();
            Temp = Sf;
            while{Temp < Sn)
            {
                ResendSegment(Sf);     //Resend a copy
                temp = temp + 1;
            }
      }
}
```

Go-Back-N ARQ

Receiver-site algorithm

```
Rn=0;
while (true)      //Repeat forever
{
    WaitForEvent();         //Sleep until an event occurs
    if(Event(ArrivalNotification))       //Data Segment arrived
    {
        ReceiveSegment();
        if ( !corruptedSegment() && seqNo == Rn)   //If expected Segment
        {
            ExtractData();
            DeliverData ( );         //Deliver data to application layer
            Rn=Rn+1;       //Slide window
            SendACKSegment(Rn);  //Send an Ack Segment
        }
    }
}
```

Efficiency of Go-Back-N ARQ

Consider the use of 10 K-bit size frames on a 10 Mbps satellite channel with 270 ms delay. What is the link utilization for go-back-N ARQ with window size of 7 assuming P = 10 -3 ?

Solution :

Channel utilization for go-back-N

$= N(1 – P) / (1 + 2a)(1-P+NP)$

P = probability of single frame error ≈ 10 -3

Channel utilization ≈ 0.01285 = 1.285%

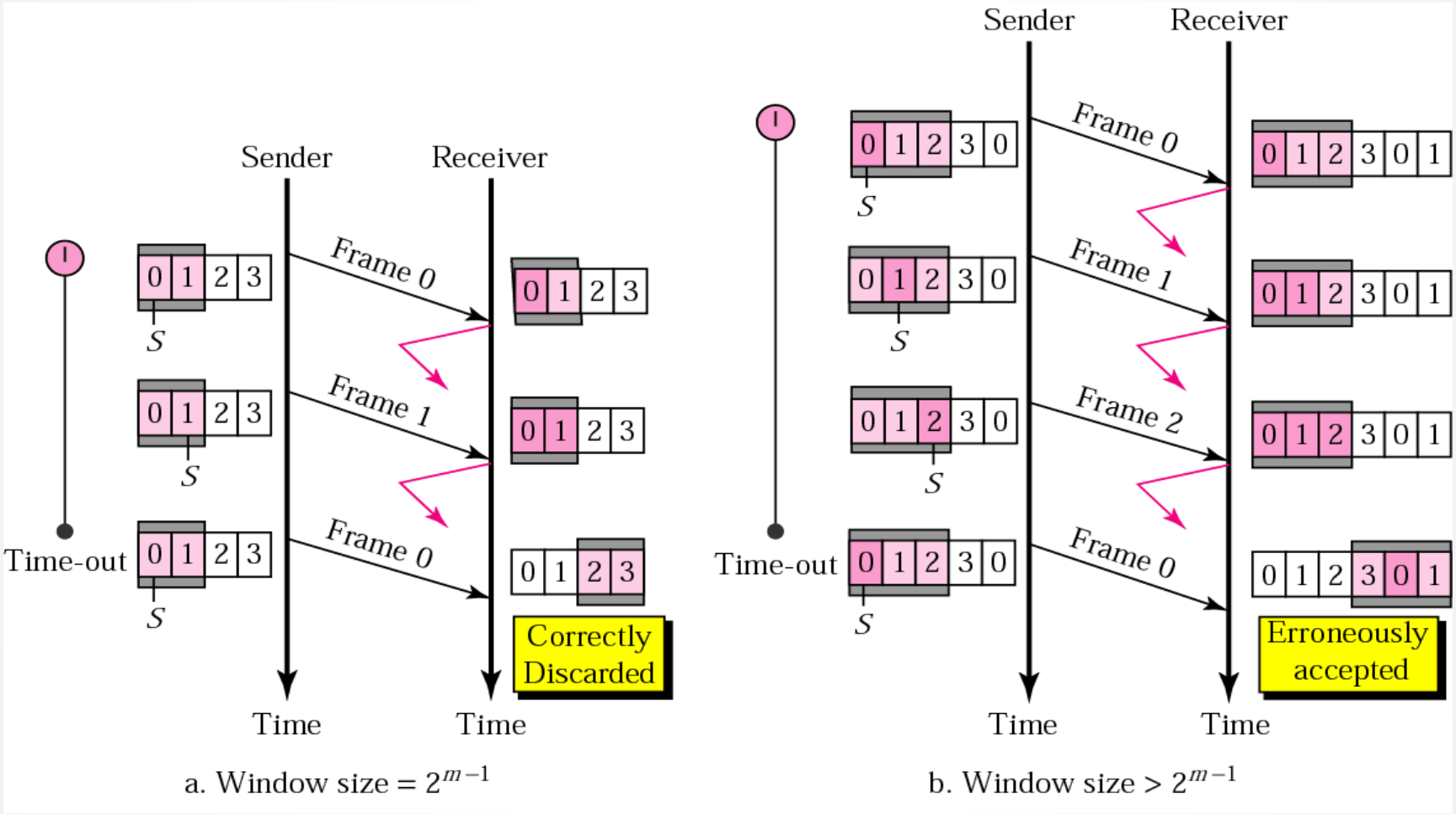
Problem with Go-Back-N ARQ

- In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames when just one frame is damaged.
- For noisy links, another mechanism called Selective RepeatARQ that does not resend N frames when just one frame is damaged; only the damaged frame is resent by handling the out of order packets received at the receiver.
- It is more efficient for noisy links, but the processing at the receiver is more complex.

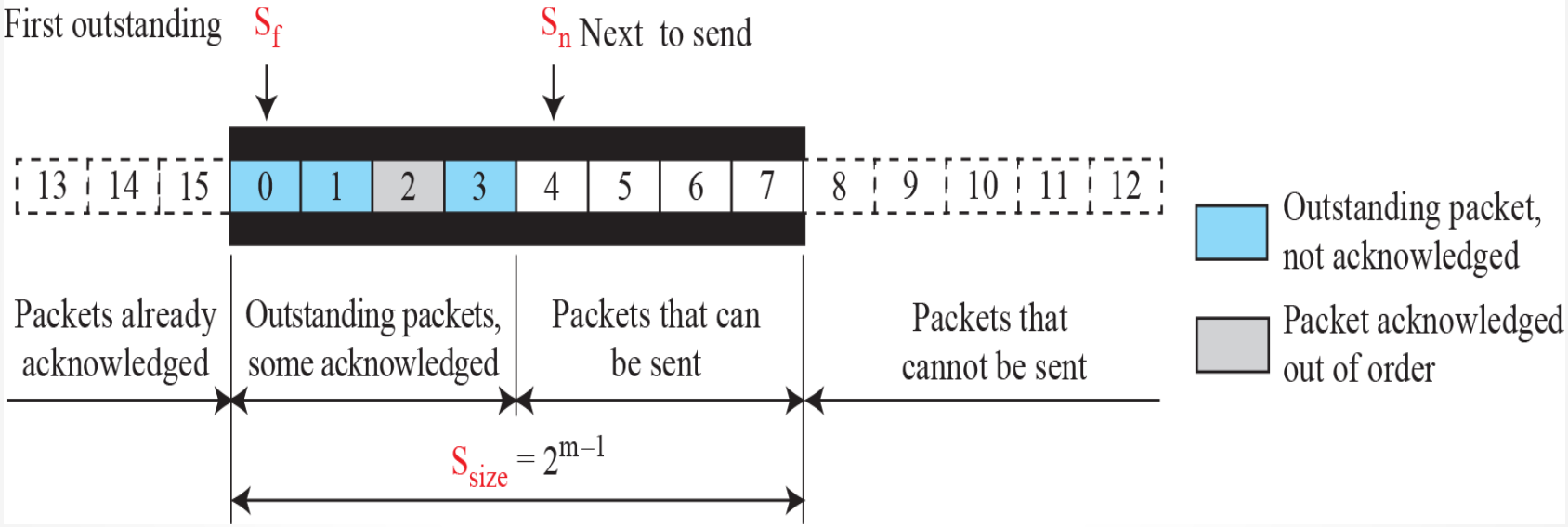
Selective Repeat ARQ

- **Sliding Window:** the range of sequence numbers that is the concern of the sender and receiver.
  - The range which is the concern of the sender is called the send sliding window. The size of the send window is 2^{m-1}.
  - The range that is the concern of the receiver is called the receive sliding window. The size of the receive window is 2^{m-1}.

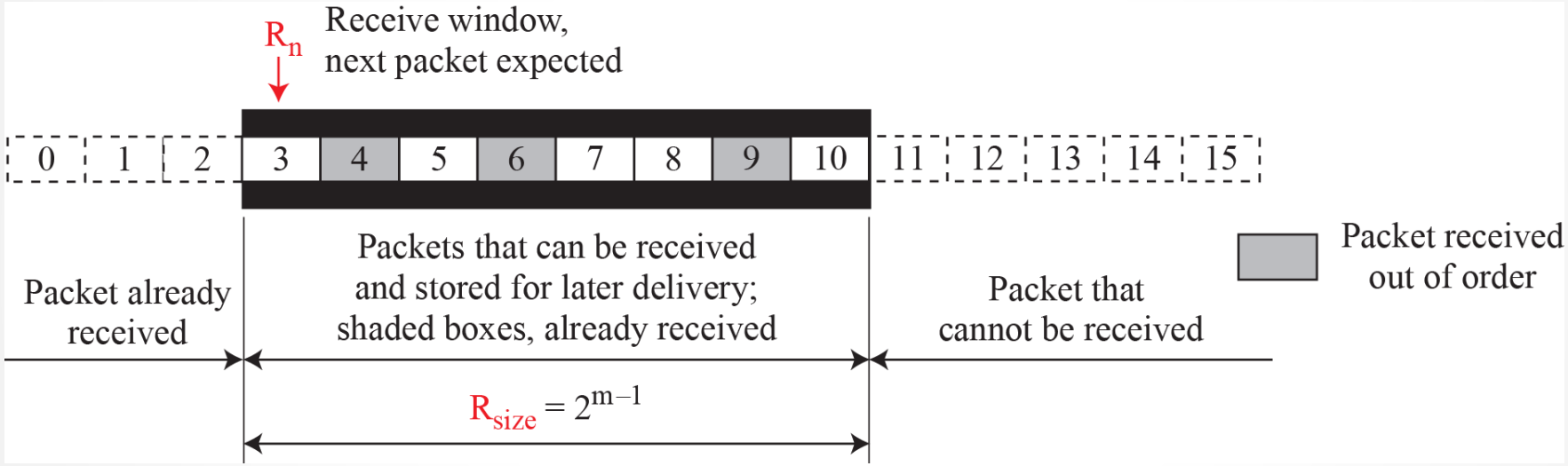
Send/Receive window size for Selective Repeat ARQ



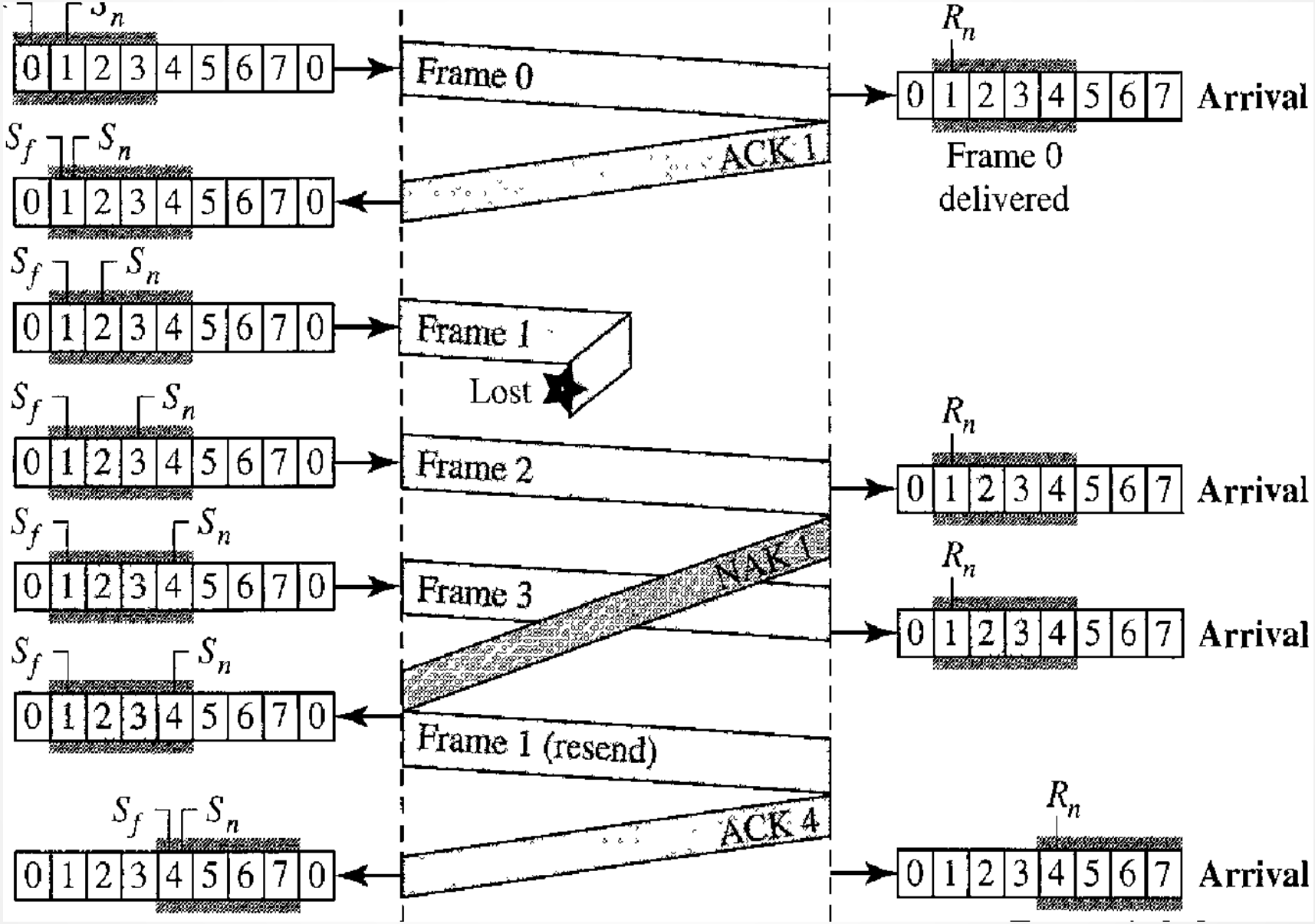
Send window for Selective Repeat ARQ



Receive window for Selective Repeat ARQ



Example of Selective Repeat ARQ



Selective Repeat ARQ

Sender-site algorithm

```
Sw = 2''' - 1;
Sn=0;
Sf = 0;                    // Frame 0 should be sent first
while (true)               //Repeat forever
   {
      WaitForEvent();        //Sleep until an event occurs
      if(Event(RequestToSend) )  //There is a packet to send
      {
            if (Sn-Sf < Sw) //If window is not full
            {
               GetData();
               MakeSegment(Sn);
               StoreSegment(Sn);     //Keep the copy
               SendSegment(Sn);      //Send the Segment
               Sn=Sn+1;
               StartTimer(Sn);
            }
      }
```

Selective Repeat ARQ

Sender-site algorithm ...

```
WaitForEvent();    //Sleep until an event occurs
  if(Event(ArrivalNotification))  //An ACK has received
  {
        ReceiveSegment(ackNo);    //Receive the ACK/NAK Segment
        if(!corruptedSegment())
        {
        if(FrameType == NAK && nackNo between Sf and Sn) {
            While(Sf <= nackNo) {
                ResendSegment(nackNo);
                StartTimer(nackNo);
            }
        }
        if(FrameType == ACK && AckNo between Sf and Sn) {
            While(Sf <  ackNo) {
                DeleteSegment(Sf);    //copy no more needed
                StopTimer(Sf);
                Sf = Sf + 1;
            }
        }
        }
  }
```

Selective Repeat ARQ

Sender-site algorithm ...

```
if (Event (TimeOut(t))          //The timer expired
        {
                StartTimer(t);
                ResendSegment(t);     //Resend a copy
                }
            }
    }
```

Selective Repeat ARQ

```
Receiver-site algorithm
    Rn=0;
    NakSent = false;
    AckNeeded = false;
    Repeat(for all slots)
        Marked (slot) = false;
    while (true)      //Repeat forever
    {
        WaitForEvent();        //Sleep until an event occurs
        if(Event(ArrivalNotification))       //Data Segment arrived
        {
            ReceiveSegment();
            if ( CorruptedSegment() && !NakSent)
            {
                SendNAK(Rn) ;
                NakSent = true;
            }
```

Selective Repeat ARQ

Receiver-site algorithm

```
if((seqNo != Rn)&& (!NakSent)){ //Not the Expected Segment
        SendNAK(Rn) ;
        NakSent = true;
}
if (seqNo in window&& !marked(seqNo)) {//Expected Segment
        StoreSegment(seqNo);
        Marked(seqNo)= true;
        whi1e(Marked(Rn) {
                DeliverData (Rn) ;
                DeleteSegment (Rn);
                Rn = Rn + 1;
                AckNeeded = true;
        }
        if (AckNeeded) {
                SendAck(Rn) ;
                AckNeeded = false;
                NakSent = false;
        }
}}}
```