Data Link Layer

Table of Contents

Data Link Layer

- Two main functions of the data link layer are:
  - ✓ Data link control
  - ✓ Media access control.
- Data link control deals with the design and procedures for communication between two adjacent nodes. It includes:
  - ✓ Framing
  - ✓ Flow control
  - ✓ Error control
- Media access control - how to share the link.

Framing

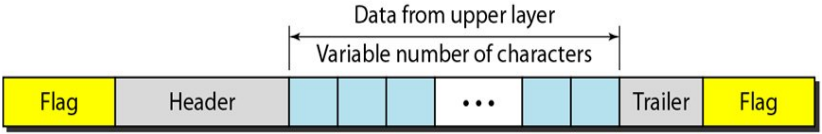
- ∝ The data link layer needs to pack bits into frames, so that each frame is distinguishable from another.
- ∝ Can we pack the whole message in one frame?
  - ✓ It can make flow and error control very inefficient.
- ∝ Framing separates a message from one source and destination, or other messages to other destinations by adding source and destination address.

Framing

- Fixed-Size Framing
  - There is no need to define the boundaries of the frames, the size it self can act as a delimiter.
- Variable-Size Framing
  - We need to define the end of the frame and the beginning of the next frame.
- Character-oriented protocol
- Bit-oriented protocol

Frame in a Character-Oriented Protocol

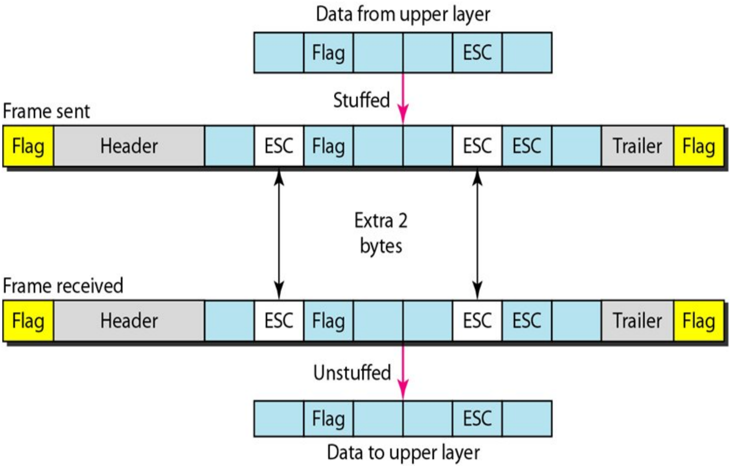
- In character-oriented protocol, data to be carried are 8-bit characters.
- Header: normally carries source & destination addresses and other control information.
- Trailer: error detection and correction redundant bits, are also multiple of 8 bits.
- Flags: used to separate one frame from the next. 1 byte flag.



Byte stuffing and unstuffing

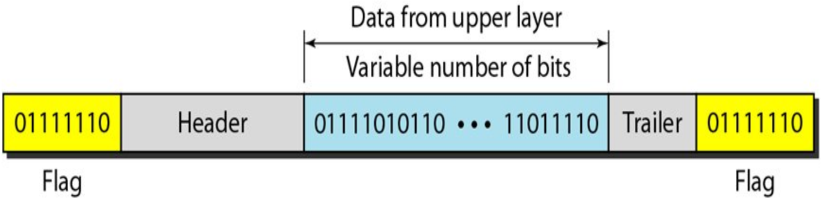
- Any pattern used for flag could also be part of the info (it can cause problem to the receiver.)
- Byte stuffing is used to fix this problem.
- In Byte stuffing, a special byte (usually called the escape character (ESC)) is added to the data section of the frame when there is a char with the same pattern as the flag.
  - ✓ On finding a ESC char, the receiver removes it from the data section and treats the next char as data.
- The ESC char that are part of the text must also be marked by another ESC char.

Byte stuffing and unstuffing



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

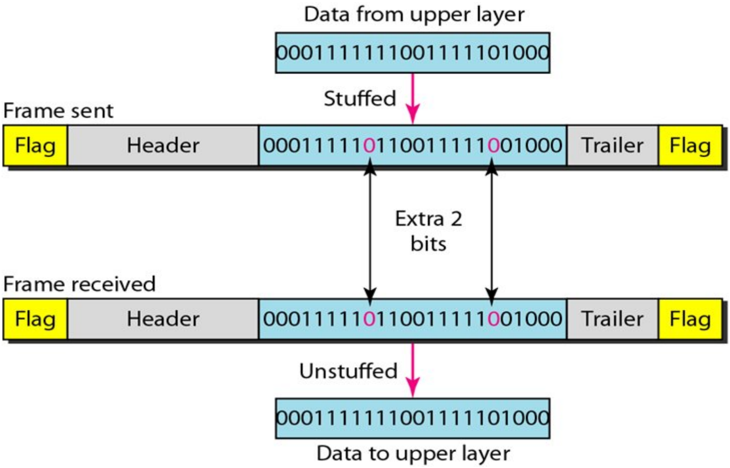
Frame in a bit-oriented protocol



Bit stuffing

- Flag: 0111110
- 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.

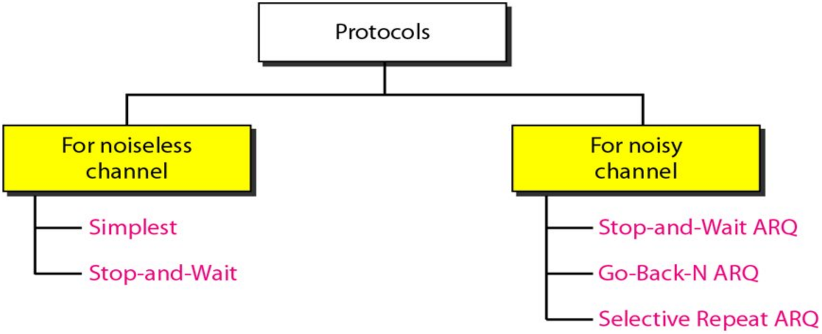
Bit stuffing



Flow and Error Control

- The most important responsibilities of the data link layer are flow control and error control. Collectively, known as data link 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.
  - ✓ The flow of data must not overwhelm the receiver.
- Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.
  - ✓ It allows the receiver to inform the sender of any frame lost or damaged in transmission.

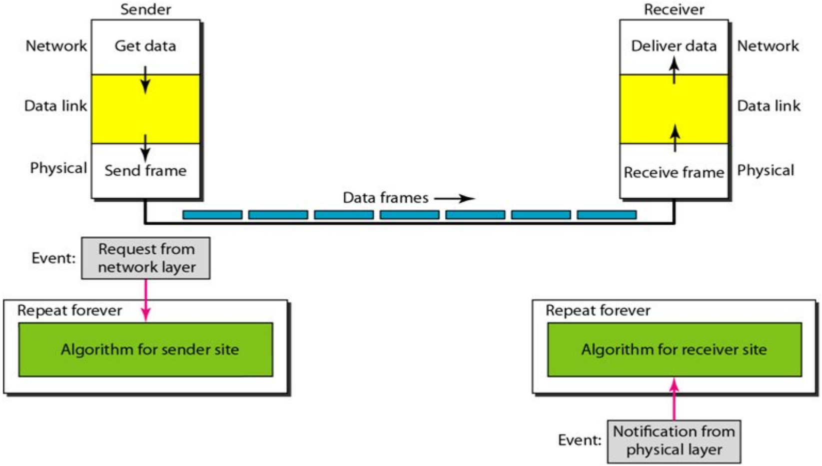
Protocols


Protocols
For noiseless channel
Simplest
Stop-and-Wait
For noisy channel
Stop-and-Wait ARQ
Go-Back-N ARQ
Selective Repeat ARQ

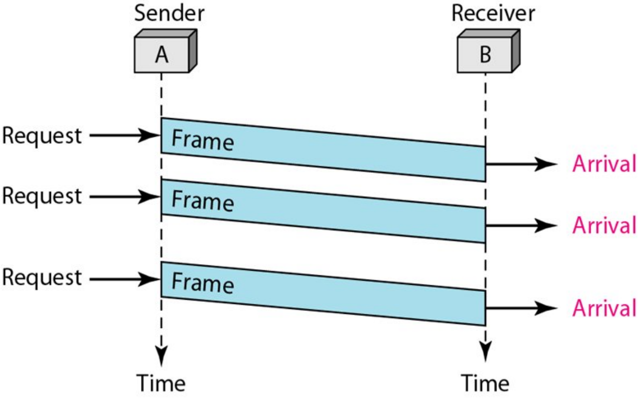
Noiseless Channel

- Noiseless Channel: An ideal channel in which no frames are lost, duplicated, or corrupted.
- Two protocols for this type of channel.
  - ✓ Simplest Protocol
  - ✓ Stop-and-Wait Protocol

Design of the simplest protocol with no flow or error control



Example of the simplest protocol with no flow or error control



Algorithm of the simplest protocol

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

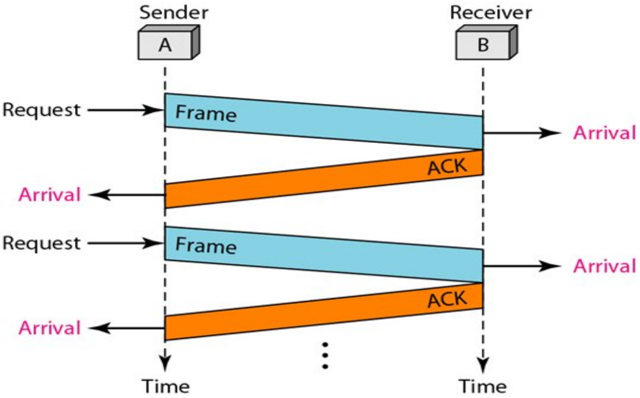
Sender-site algorithm for the simplest protocol

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

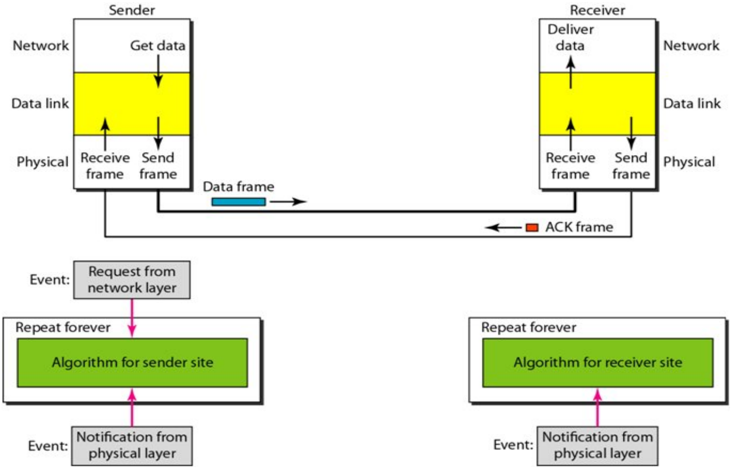
Receiver-site algorithm for the simplest protocol

Stop-and-Wait Protocol

- In this protocol, the sender sends one frame, stops until it receives confirmation from the receiver and then sends the next frame.



Design of Stop-and-Wait Protocol



Sender-Site Algorithm for Stop-and-Wait Protocol

```
while(true)                          //Repeat forever
canSend = true                       //Allow the first frame to go
{
 WaitForEvent();                     // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
     GetData();
     MakeFrame();
     SendFrame();                    //Send the data frame
     canSend = false;                //Cannot send until ACK arrives
  }
 WaitForEvent();                     // Sleep until an event occurs
  if(Event(ArrivalNotification)  // An ACK has arrived
   {
     ReceiveFrame();                 //Receive the ACK frame
     canSend = true;
   }
}
```

Receiver-Site Algorithm for Stop-and-Wait Protocol

```
while(true)                          //Repeat forever
{
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(ArrivalNotification))     //Data frame arrives
  {
     ReceiveFrame();
     ExtractData();
     Deliver(data);                  //Deliver data to network layer
     SendFrame();                    //Send an ACK frame
  }
}
```

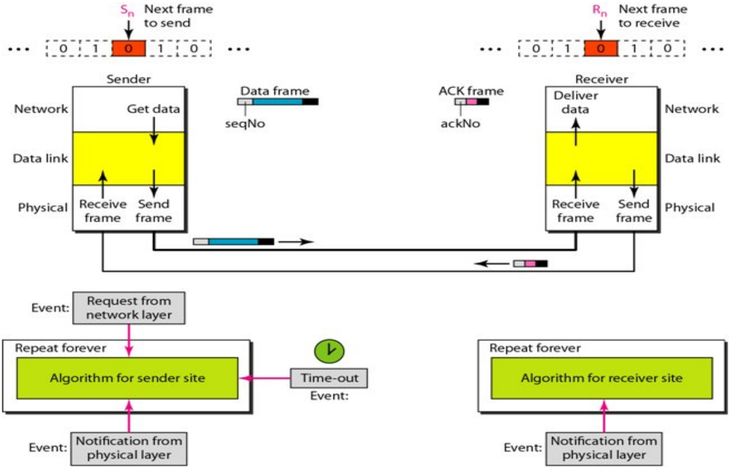
Noisy Channel

- We discuss three protocols in this section that use error control.
  - Stop-and-Wait Automatic Repeat Request (Stop-and-Wait ARQ)
  - Go-Back-N ARQ
  - Selective Repeat ARQ

Stop-and-Wait ARQ

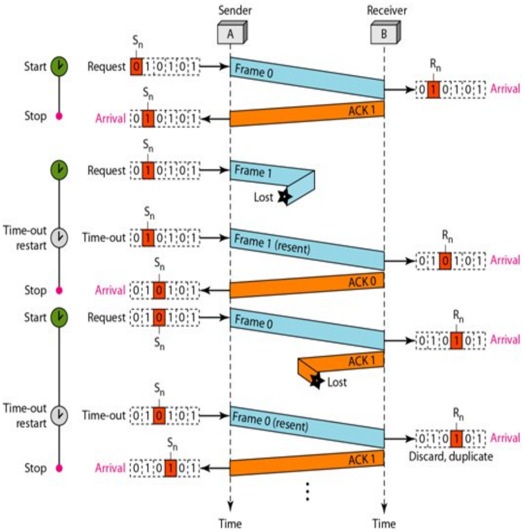
- Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.
- In Stop-and-Wait ARQ, we use sequence numbers to number the frames.
- The sequence numbers are based on modulo-2 arithmetic.
- In Stop-and-Wait ARQ, the acknowledgment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

Design of the Stop-and-Wait ARQ Protocol



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.



Sender-site algorithm for Stop-and-Wait ARQ

```
Sn = 0;                          // Frame 0 should be sent first
canSend = true;                  // Allow the first request to go
while(true)                      // Repeat forever
{
  WaitForEvent();                // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
     GetData();
     MakeFrame(Sn);                        //The seqNo is Sn
     StoreFrame(Sn);                       //Keep copy
     SendFrame(Sn);
     StartTimer();
     Sn = Sn + 1;
     canSend = false;
  }
  WaitForEvent();                          // Sleep
```

(continued...)

Sender-site algorithm for Stop-and-Wait ARQ

```
    if(Event(ArrivalNotification)      // An ACK has arrived
    {
      ReceiveFrame(ackNo);             //Receive the ACK frame
      if(not corrupted AND ackNo == S_n) //Valid ACK
        {
          Stoptimer();
          PurgeFrame(S_{n-1});         //Copy is not needed
          canSend = true;
        }
     }

    if(Event(TimeOut)                  // The timer expired
    {
     StartTimer();
     ResendFrame(S_{n-1});             //Resend a copy check
    }
}
```

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

```
R_n = 0;                        // Frame 0 expected to arrive first
while(true)
{
  WaitForEvent();               // Sleep until an event occurs
  if(Event(ArrivalNotification)) //Data frame arrives
  {
     ReceiveFrame();
     if(corrupted(frame));
        sleep();
     if(seqNo == R_n)            //Valid data frame
     {
      ExtractData();
       DeliverData();            //Deliver data
       R_n = R_n + 1;
     }
      SendFrame(R_n);            //Send an ACK
  }
}
```

Stop-and-Wait ARQ

- Problem: 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

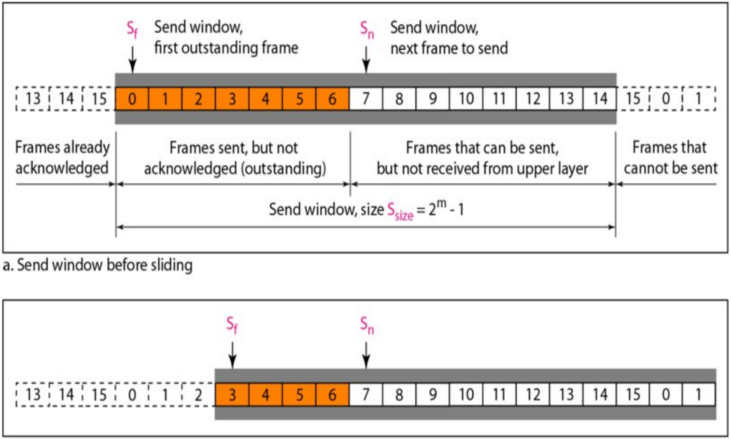
$$(1 \times 10^{6}) \times (20 \times 10^{-3}) = 20,000 \text{ bits}$$

- The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and back again.
- However, the system sends only 1000 bits.
- So, 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.

Go-Back-N ARQ

- In Go-Back-N ARQ protocol, the sequence number 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}.
  - ✓ S_f: First outstanding frame for which acknowledgment is not yet received.
  - ✓ S_n: Next Frame to send
  - ✓ S_{size}: Send Window size, $= 2^m - 1$
- The send window can slide one or more slots when a valid acknowledgment arrives.

Send window for Go-Back-N ARQ

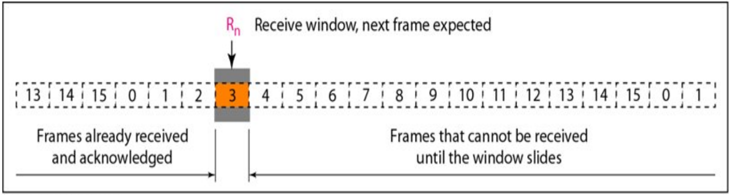


a. Send window before sliding

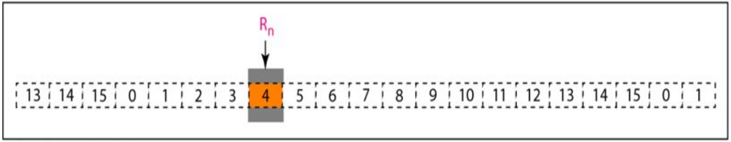
b. Send window after sliding

Receive window for Go-Back-N ARQ

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



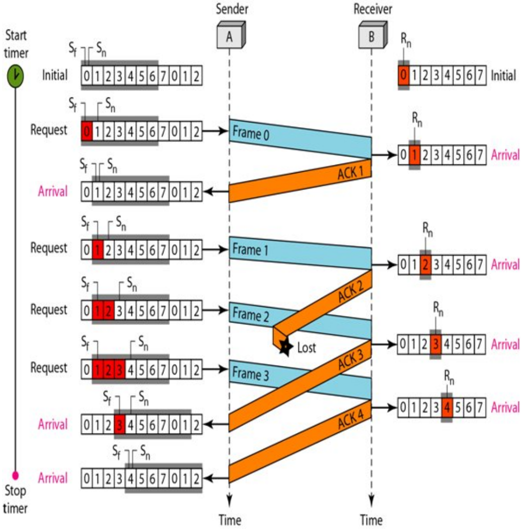
a. Receive window



b. Window after sliding

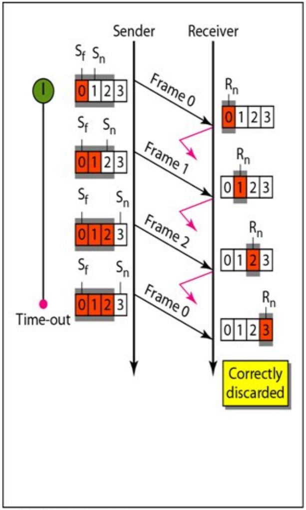
Example of Go-Back-N

- The example shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.
- Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.
- Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

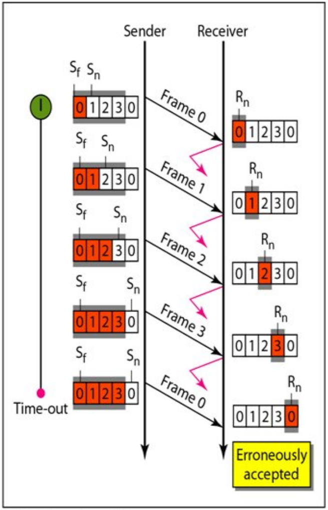


Window size for Go-Back-N ARQ

- In Go-Back-N ARQ, the size of the send window must be less than 2^m.
- The Size of the receive window is always 1.

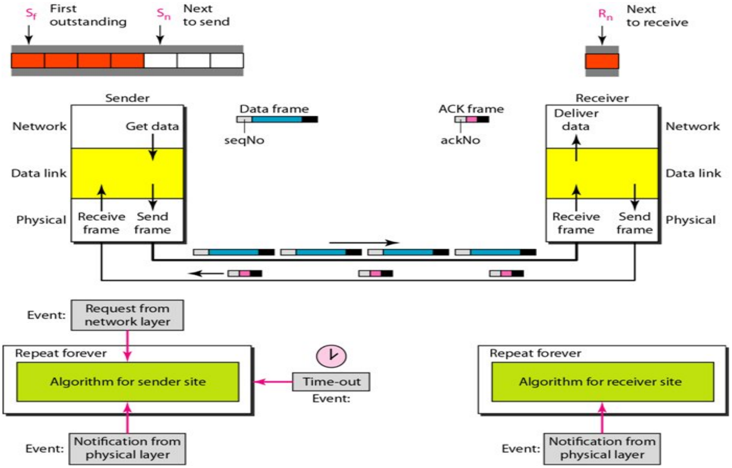


a. Window size $< 2^m$



b. Window size $= 2^m$

Design of Go-Back-N ARQ



Go-Back-N sender algorithm

```
S_w = 2^m - 1;
S_f = 0;
S_n = 0;

while (true)                           //Repeat forever
{
 WaitForEvent();
  if(Event(RequestToSend))             //A packet to send
  {
     if(S_n-S_f >= S_w)                //If window is full
           Sleep();
     GetData();
     MakeFrame(S_n);
     StoreFrame(S_n);
     SendFrame(S_n);
     S_n = S_n + 1;
     if(timer not running)
          StartTimer();
  }

```

continued...

Go-Back-N sender algorithm

```
   if(Event(ArrivalNotification)) //ACK arrives
   {
      Receive(ACK);
      if(corrupted(ACK))
           Sleep();
      if((ackNo>S_f)&&(ackNo<=S_n))  //If a valid ACK
      While(S_f <= ackNo)
       {
        PurgeFrame(S_f);
        S_f = S_f + 1;
       }
       StopTimer();
   }

   if(Event(TimeOut))                //The timer expires
   {
    StartTimer();
    Temp = S_f;
    while(Temp < S_n);
     {
      SendFrame(S_f);
      S_f = S_f + 1;
     }
   }
}
```

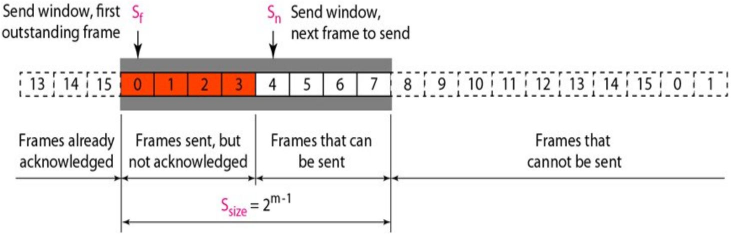
Go-Back-N receiver algorithm

```
Rn = 0;

while (true)                           //Repeat forever
{
  WaitForEvent();

  if(Event(ArrivalNotification))  /Data frame arrives
  {
     Receive(Frame);
     if(corrupted(Frame))
          Sleep();
     if(seqNo == Rn)                   //If expected frame
     {
       DeliverData();                  //Deliver data
       Rn = Rn + 1;                    //Slide window
       SendACK(Rn);
     }
  }
}
```

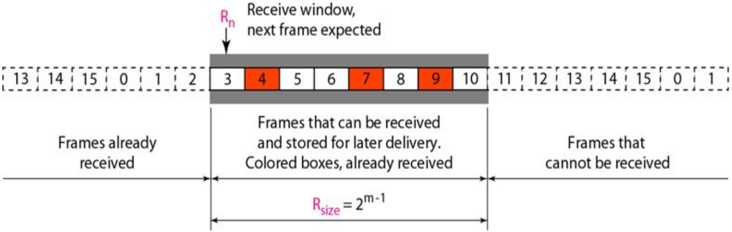
Selective Repeat ARQ

- Selective Repeat ARQ is more efficient for noisy channels.
  - It does not require to resend N frames when just one frame is damaged (as in Go-Back-N ARQ).
  - Only the damaged frames are resent.
  - Processing at the receiver is more complex.
- Windows:
  - Send Window: 2^{m-1}
  - Receive Window: 2^{m-1}
  - E.g. if $m = 4$, the sequence number go from 0 ... 15, but the size of the window is 8

Window for Selective Repeat ARQ



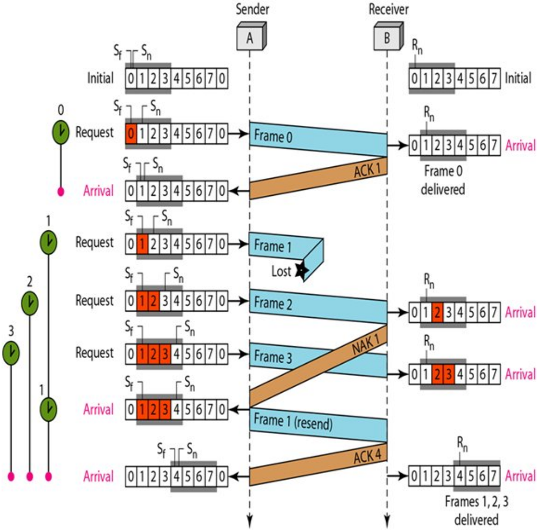
Send window for Selective Repeat ARQ



Receive window for Selective Repeat ARQ

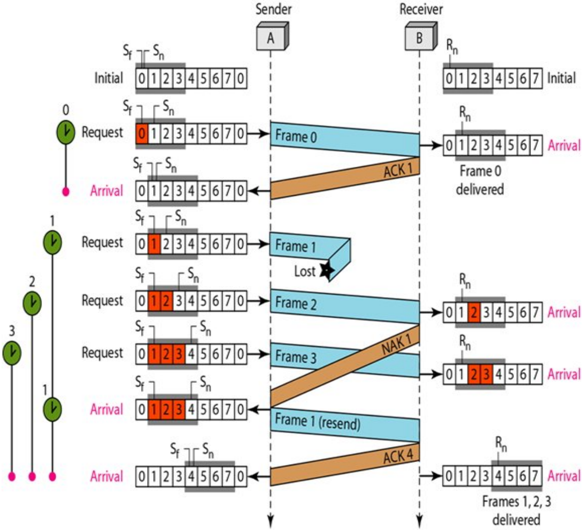
Example of Selective Repeat ARQ

- In this example frame 1 is lost.
- 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.



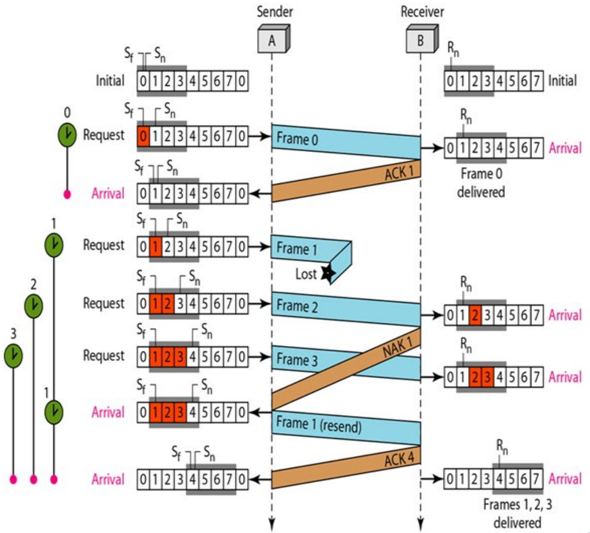
Example of Selective Repeat ARQ (continued)

- At the receiver site, frame 2 arrives before frame 1. It is stored and marked, but it cannot be delivered to the network layer 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.

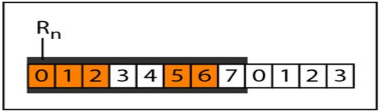


Example of Selective Repeat ARQ (continued)

- 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 protocol does not want to crowd the network with unnecessary NAKs and resent frames.
- 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.



Delivery of data in Selective Repeat ARQ

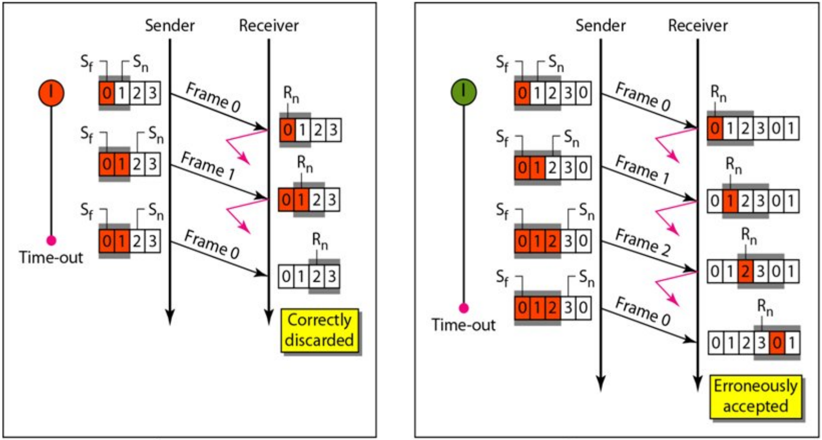


a. Before delivery

b. After delivery

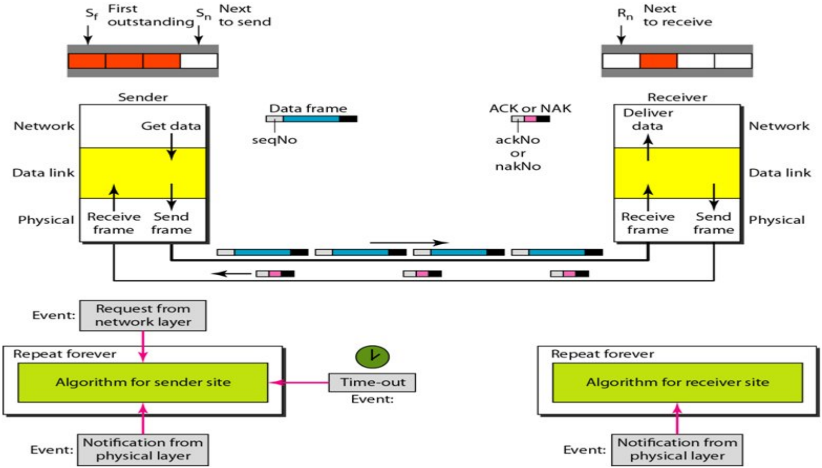
Selective Repeat ARQ, Window Size

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



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

Design of Selective Repeat ARQ



Sender-site Selective Repeat algorithm

```
Sw = 2^(m-1) ;
Sf = 0;
Sn = 0;

while (true)                          //Repeat forever
{
  WaitForEvent();
  if(Event(RequestToSend))            //There is a packet to send
  {
     if(Sn-Sf >= Sw)                  //If window is full
           Sleep();
     GetData();
     MakeFrame(Sn);
     StoreFrame(Sn);
     SendFrame(Sn);
     Sn = Sn + 1;
     StartTimer(Sn);
  }

```

Continued...

```
if(Event(ArrivalNotification)) //ACK arrives
{
   Receive(frame);             //Receive ACK or NAK
   if(corrupted(frame))
       Sleep();
   if (FrameType == NAK)
      if (nakNo between Sf and Sn)
      {
       resend(nakNo);
       StartTimer(nakNo);
      }
   if (FrameType == ACK)
      if (ackNo between Sf and Sn)
      {
        while(sf < ackNo)
        {
         Purge(sf);
         StopTimer(sf);
         Sf = Sf + 1;
        }
      }
}

if(Event(TimeOut(t)))          //The timer expires
{
 StartTimer(t);
 SendFrame(t);
}
}
```

Receiver-site Selective Repeat algorithm

```
Rn = 0;
NakSent = false;
AckNeeded = false;
Repeat(for all slots)
    Marked(slot) = false;

while (true)                                //Repeat forever
{
  WaitForEvent();

  if(Event(ArrivalNotification))            /Data frame arrives
  {
     Receive(Frame);
     if(corrupted(Frame))&& (NOT NakSent)
     {
      SendNAK(Rn);
      NakSent = true;
      Sleep();
     }
     if(seqNo <> Rn)&& (NOT NakSent)
     {
      SendNAK(Rn);
```

Continued...

Receiver-site Selective Repeat algorithm

```
        NakSent = true;
        if ((seqNo in window)&&(!Marked(seqNo))
        {
         StoreFrame(seqNo)
         Marked(seqNo)= true;
         while(Marked(R_n))
          {
           DeliverData(R_n);
           Purge(R_n);
           R_n = R_n + 1;
           AckNeeded = true;
          }
           if(AckNeeded);
           {
           SendAck(R_n);
           AckNeeded = false;
           NakSent = false;
           }
        }
      }
  }
}
```