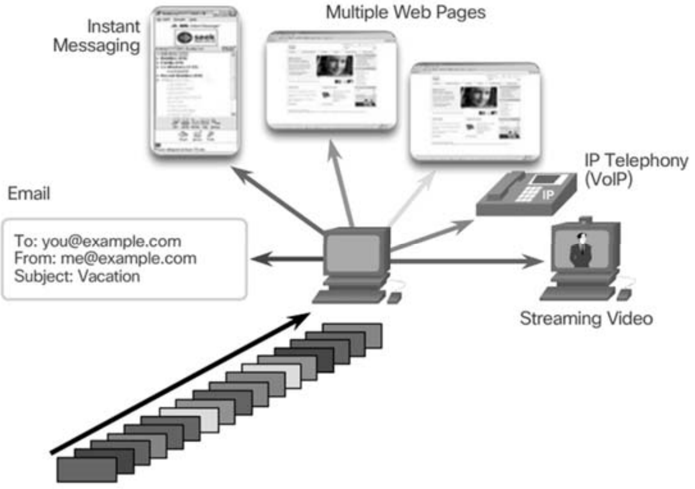
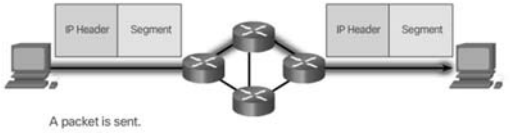
Data Link Layer

Department of Computer Engineering
Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang



Connectionless Communication



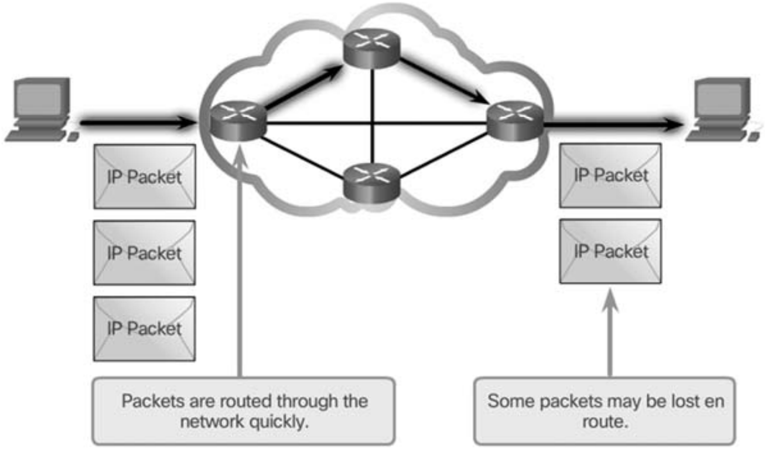
The sender doesn't know:

- If the receiver is present
- If the packet arrived
- If the receiver can read the packet

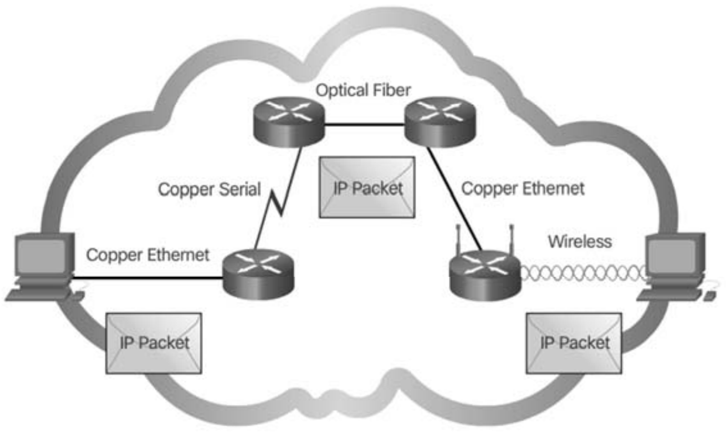
The receiver doesn't know:

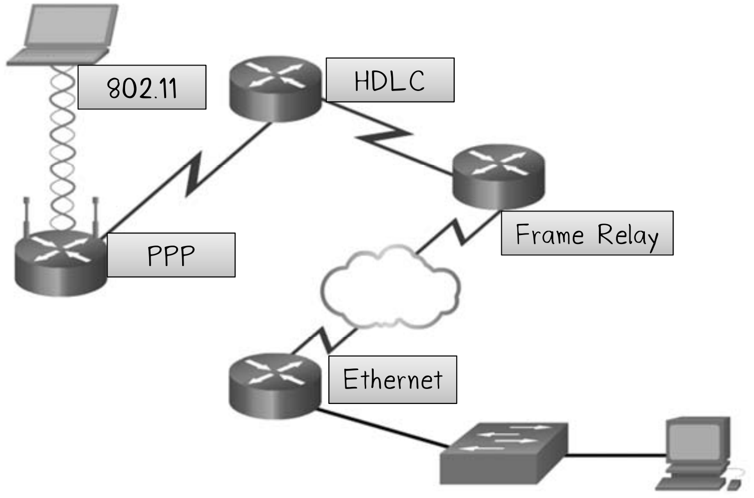
- When it is coming

Best Effort Process

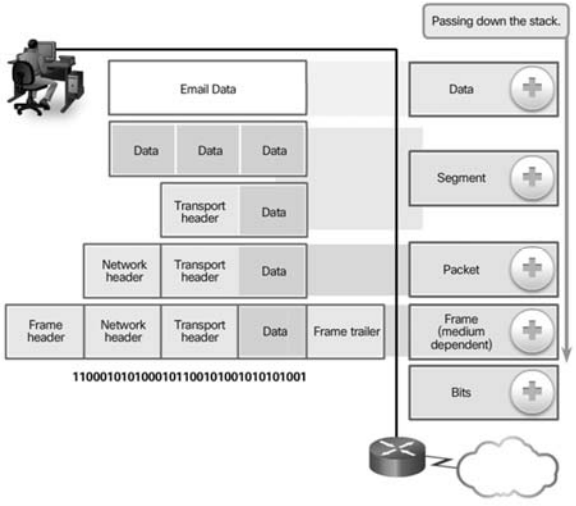

IP Packet
IP Packet
IP Packet
IP Packet
IP Packet
Packets are routed through the network quickly.
Some packets may be lost en route.

Media Independent Process

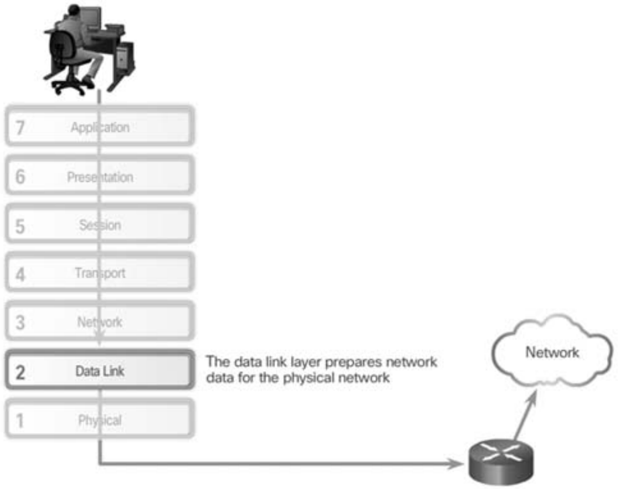




Protocol Data Units

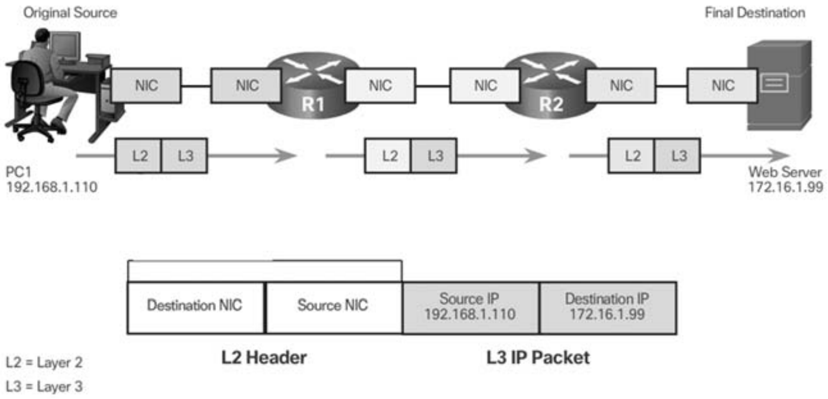


Data Link Layer

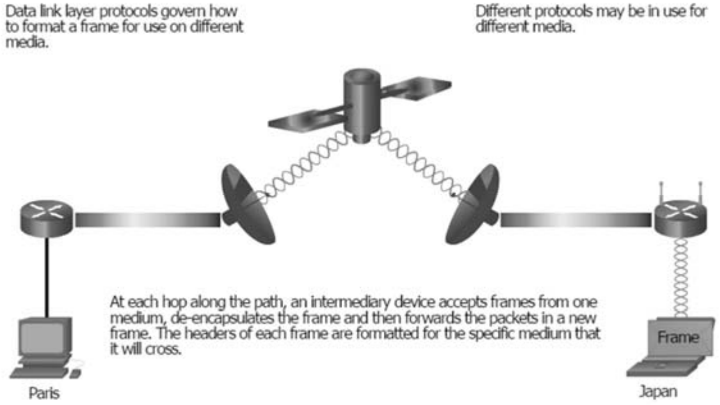


The data link layer prepares network data for the physical network

Data Link Layer



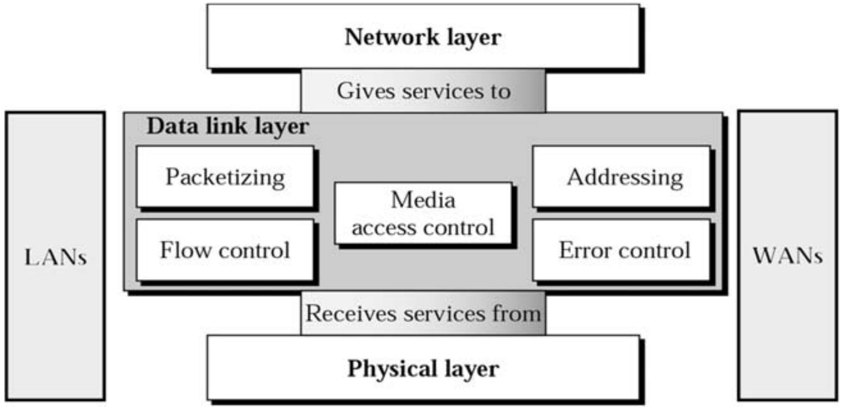
Data Link Layer


Data link layer protocols govern how to format a frame for use on different media.

Different protocols may be in use for different media.

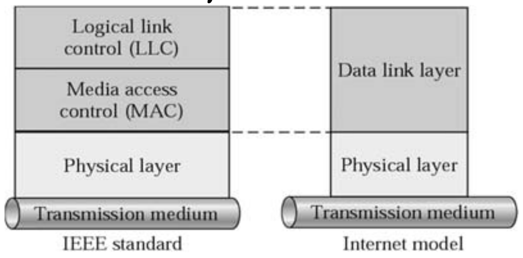
At each hop along the path, an intermediary device accepts frames from one medium, de-encapsulates the frame and then forwards the packets in a new frame. The headers of each frame are formatted for the specific medium that it will cross.

Position of the data-link layer

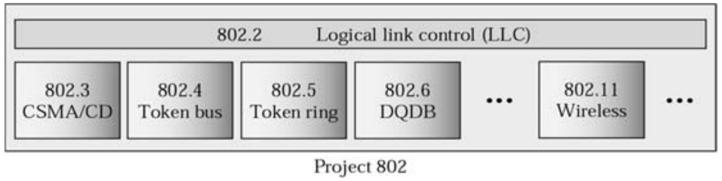


Sublayers

- LLC and MAC sublayers



- IEEE standards for LANs



Chapter 11
Data Link Control

Jirasak Sittigorn
Department of Computer Engineering
Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang

Data Link Layer

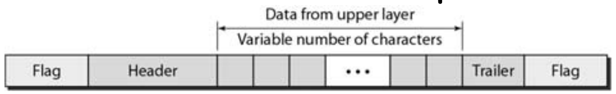
- Main function
  - Data link control : node-to-node comm.
    - Framing
    - Flow control
    - Error control
    - Software-implemented protocols
  - Media access control : Share link comm.

Framing

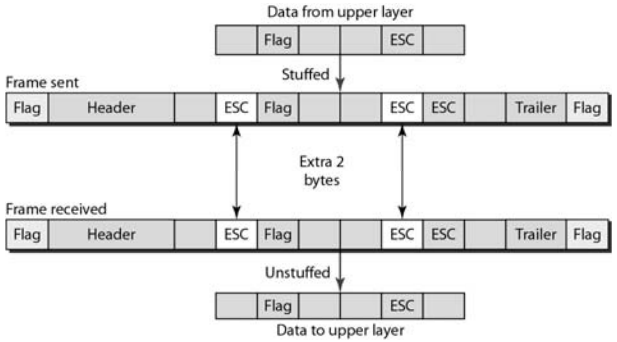
- Data Link Layer (bits) <-> Physical Layer (Signal)
- Physical Layer
  - Bit Synchronization (bit duration & timing)
- Data Link Layer
  - Addressing (Destination & Source)
  - Flow control & Error control
  - Framing (Size)
    - Fixed-Size Framing
    - Variable-Size Framing

Character-oriented protocol

- A frame in a character-oriented protocol

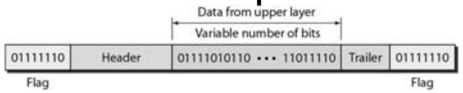


- Byte stuffing and unstuffing

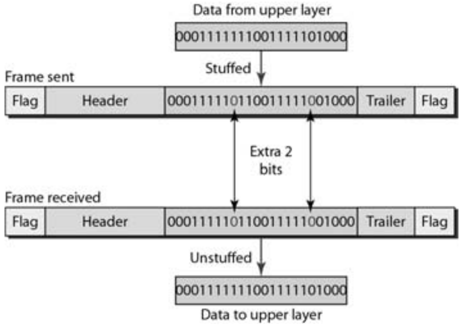


Bit-Oriented Protocol

- A frame in a bit-oriented protocol



- Bit stuffing and unstuffing



B. A. Forouzan, Data Communications and Networking, 4th edition, McGRAW-HILL

Flow & Error control

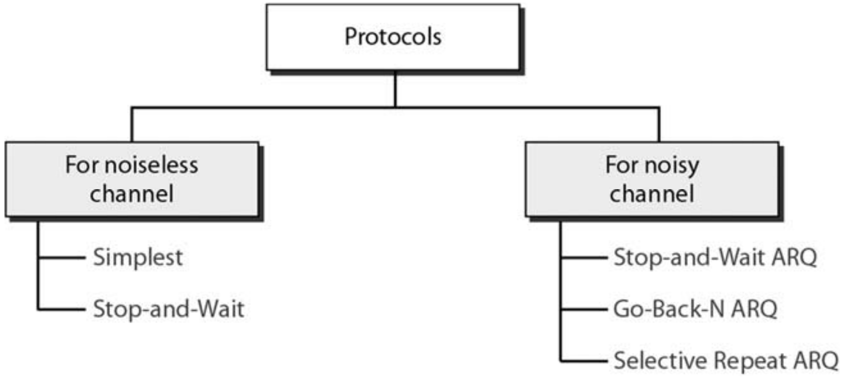
- Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
- Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.

B. A. Forouzan, Data Communications and Networking, 4th edition, McGRAW-HILL

Protocols (Activity)

- ให้ นศ. ออกแบบขั้นตอนในการรับส่งข้อมูล แล้วแสดงลำดับการส่งข้อมูล

B. A. Forouzan, Data Communications and Networking, 4th edition, McGRAW-HILL



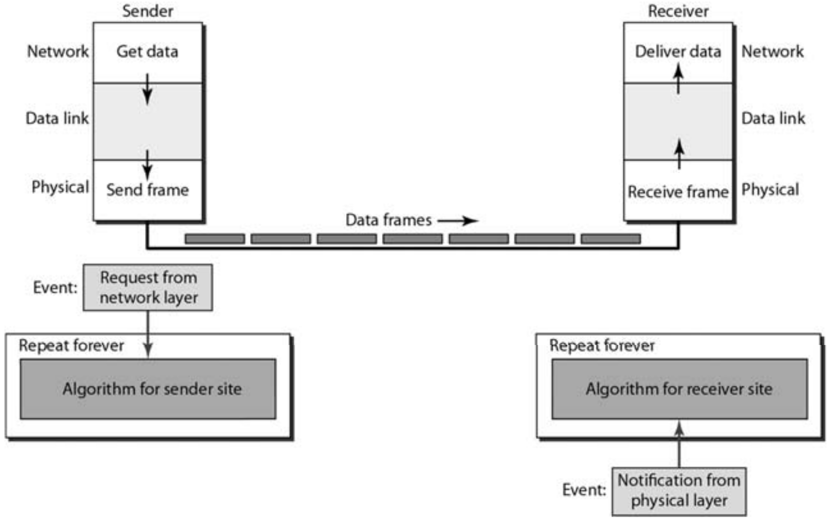
Protocols

For noiseless channel
- Simplest
- Stop-and-Wait

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

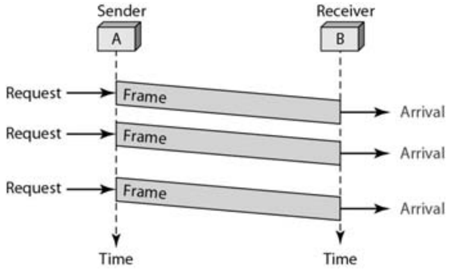
B. A. Forouzan, Data Communications and Networking, 4th edition, McGRAW-HILL

Simplest Protocol

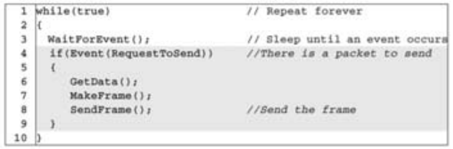


Simplest Protocol

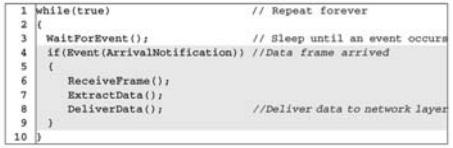
- Flow diagram



- Algorithm

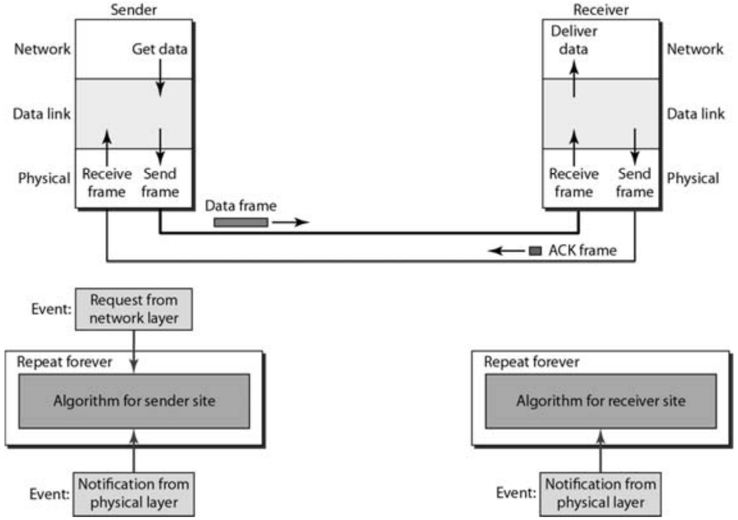


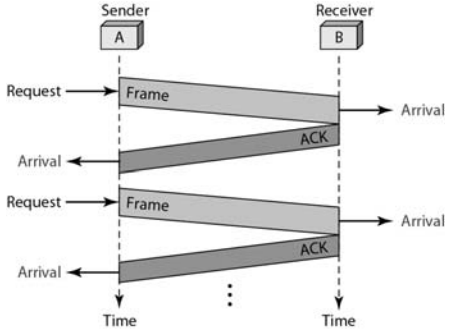
```
while(true)                       // Repeat forever
{
  WaitForEvent();                 // Sleep until an event occurs
   if(Event(RequestToSend))       //There is a packet to send
   {
       GetData();
       MakeFrame();
       SendFrame();               //Send the frame
   }
}
```

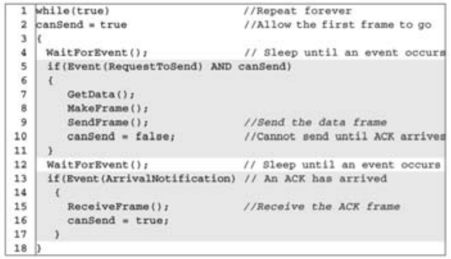


```
while(true)                          // Repeat forever
{
  WaitForEvent();                    // Sleep until an event occurs
   if(Event(ArrivalNotification))    //Data frame arrived
   {
       ReceiveFrame();
       ExtractData();
       DeliverData();                //Deliver data to network layer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

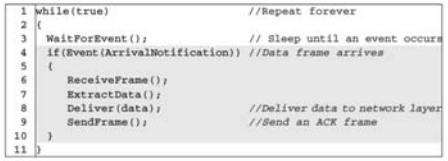
Stop-and-Wait Protocol







```
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;
   }
}
```

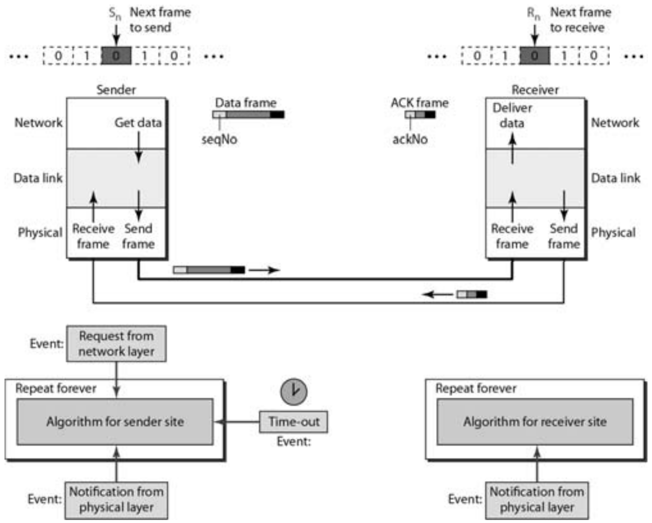


```
while(true)                          //Repeat forever
{
  WaitForEvent();                    // Sleep until an event occurs
   if(Event(ArrivalNotification))    //Data frame arrives
   {
       ReceiveFrame();
       ExtractData();
       Deliver(data);                //Deliver data to network layer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

Stop-and-Wait Automatic Repeat Request

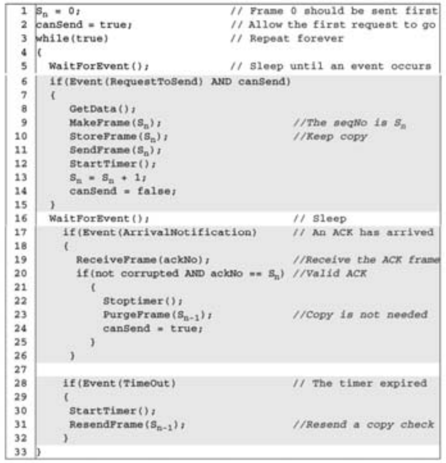
- copy & retransmitting frame
  - when the timer expires => Error correction
- frame sequence numbers
  - based on modulo-2 arithmetic
- the acknowledgment number
  - modulo-2 arithmetic the sequence number of the next frame expected

Stop-and-Wait Automatic Repeat Request

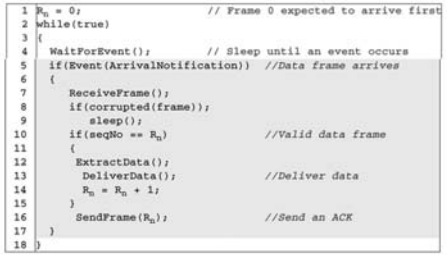


Stop-and-Wait Automatic Repeat Request

- Algorithm



```
S_n = 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(S_n);                        //The seqNo is S_n
     StoreFrame(S_n);                       //Keep copy
     SendFrame(S_n);
     StartTimer();
     S_n = S_n + 1;
     canSend = false;
  }
  WaitForEvent();                           // Sleep
    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
    }
}
```

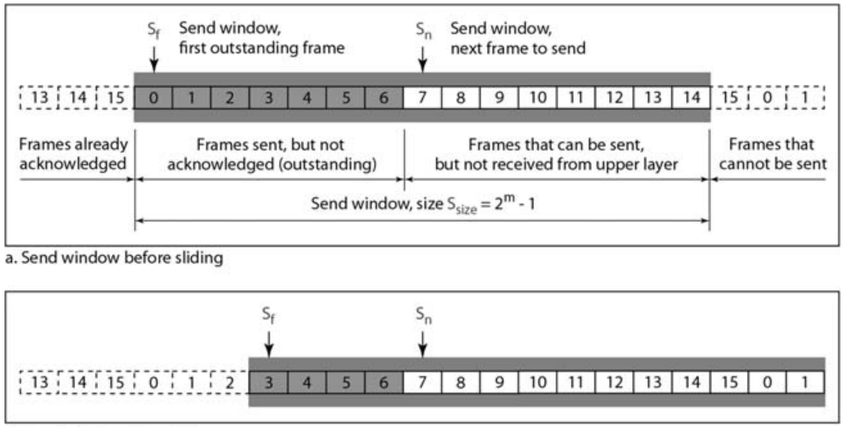


```
R_n = 0;                       // Frame 0 expected to arrive first
while(true)
{
  WaitForEvent();              // Sleep until an event occurs
  if(Event(ArrivalNotification))  //Data frame arrives
  {
     ReceiveFrame();
     if(corrupted(frame));
        sleep();
     if(seqNo == R_n)        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

Stop-and-Wait Automatic Repeat Request

- Flow diagram
  - Send 5
  - Data 2nd lost
  - ACK 3th lost

Go-Back-N Automatic Repeat Request



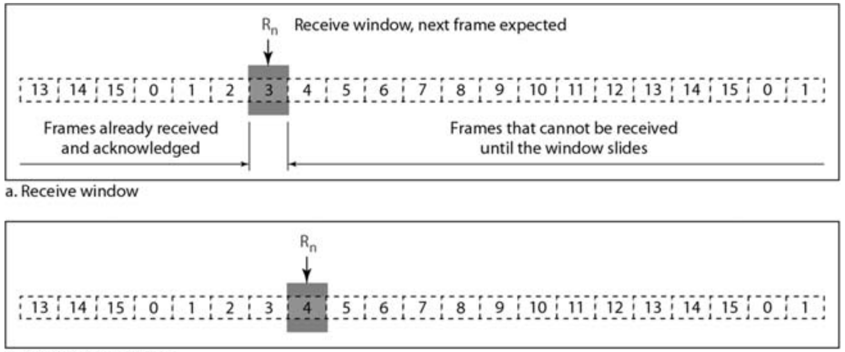
S_f Send window, first outstanding frame
S_n Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged (outstanding)
Frames that can be sent, but not received from upper layer
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$
S_f
S_n
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1

a. Send window before sliding

b. Send window after sliding

- the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

Go-Back-N Automatic Repeat Request

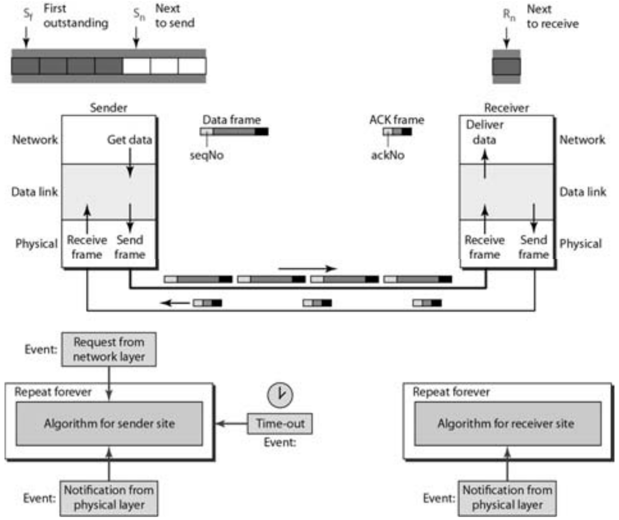


R_n Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received and acknowledged
Frames that cannot be received until the window slides
R_n
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1

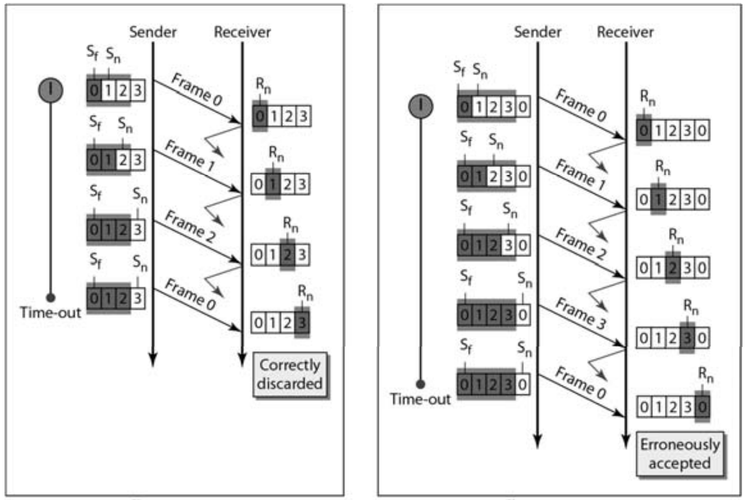
a. Receive window

b. Window after sliding

Go-Back-N Automatic Repeat Request

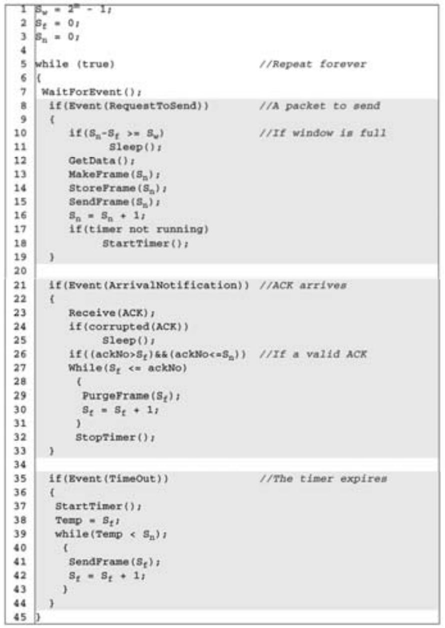


Go-Back-N Automatic Repeat Request

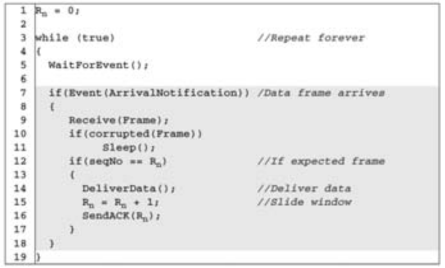


a. Window size < 2^m

b. Window size = 2^m



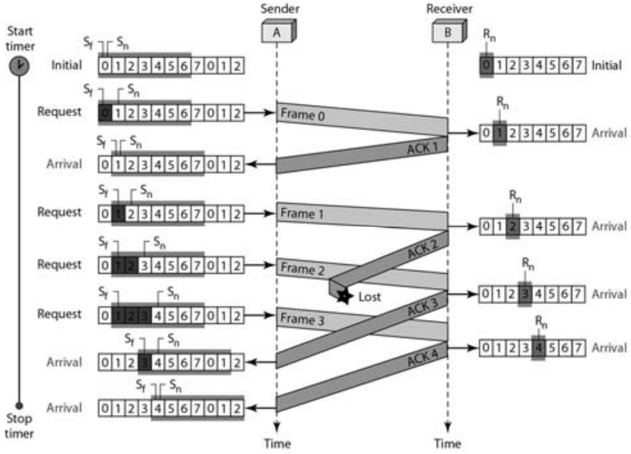
```
S_w = 2^m - 1;
S_f = 0;
S_n = 0;

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();
  }

  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;
    }
  }
}
```



```
R_n = 0;

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

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

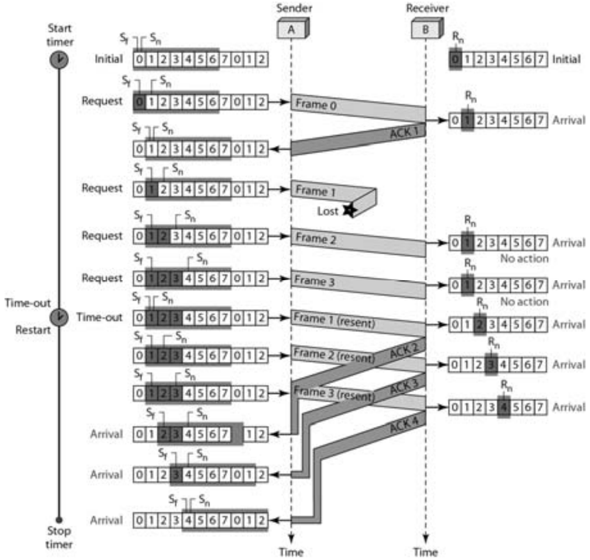
Go-Back-N Automatic Repeat Request

- Flow diagram



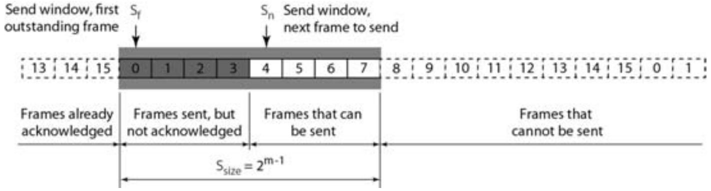
Go-Back-N Automatic Repeat Request

- Flow diagram

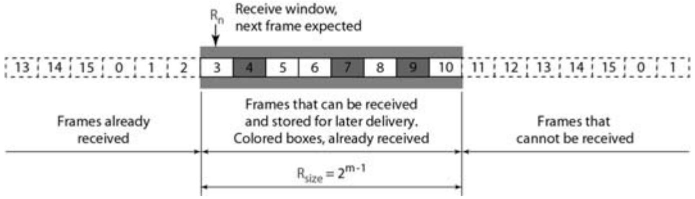


Selective Repeat Automatic Repeat Request

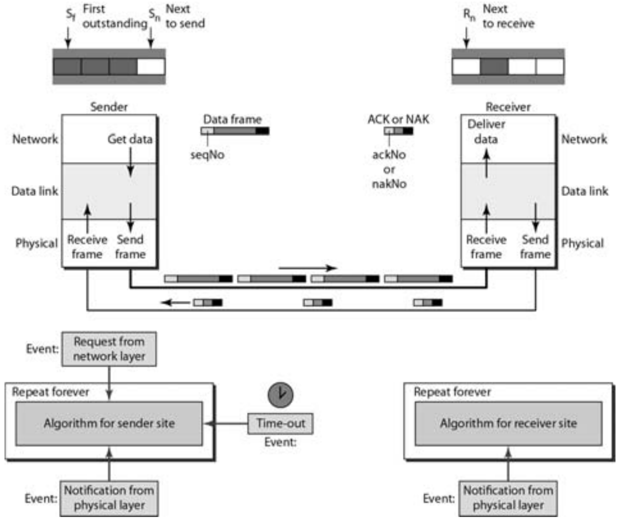
- Send window


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
$S_{size} = 2^{m-1}$

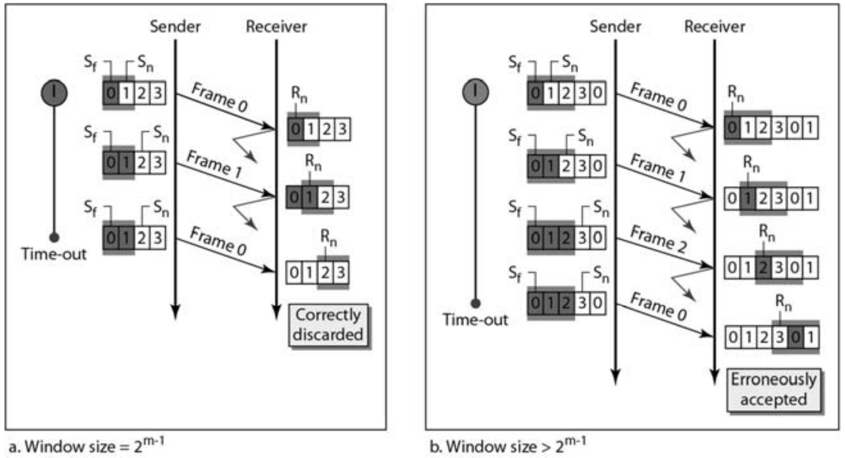
- Receive window


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
$R_{size} = 2^{m-1}$

Selective Repeat Automatic Repeat Request

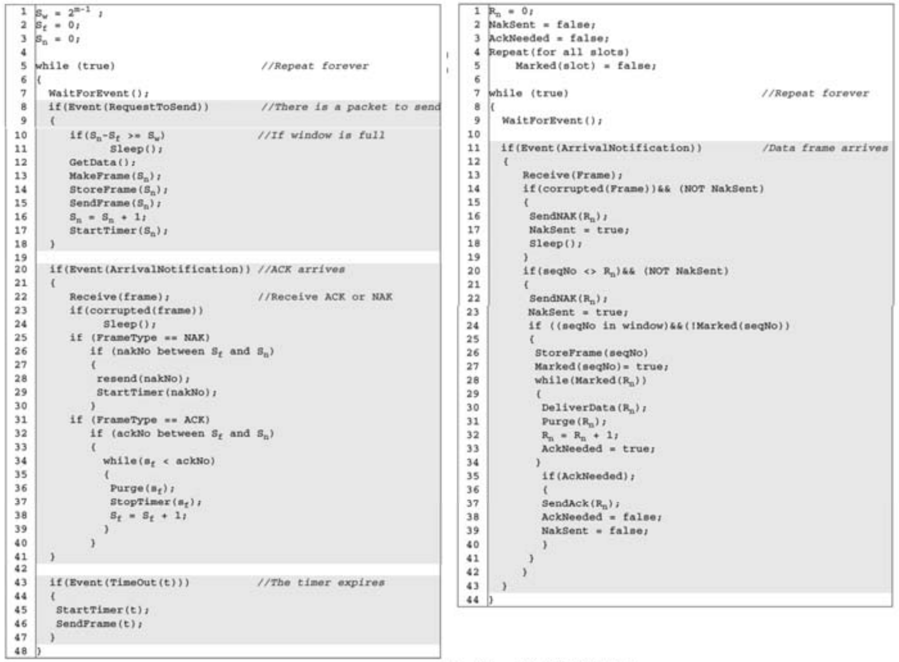


Selective Repeat Automatic Repeat Request



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

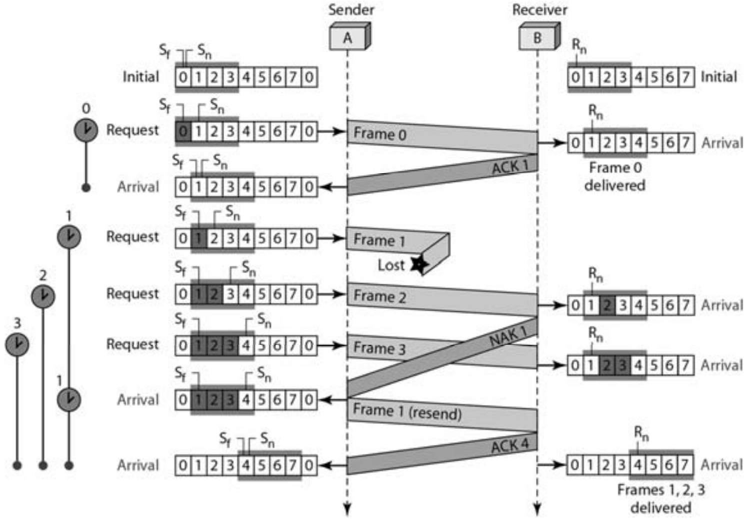
b. Window size > 2^{m-1}



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

while (true)                            //Repeat forever
{
  WaitForEvent();
  if(Event(RequestToSend))              //There is 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;
     StartTimer(S_n);
  }

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

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

```
R_n = 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(R_n);
      NakSent = true;
      Sleep();
     }
     if(seqNo <> R_n)&& (NOT NakSent)
     {
      SendNAK(R_n);
      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;
        }
      }
     }
  }
}
```

Selective Repeat Automatic Repeat Request



Sender
Receiver
A
B
S_f
S_n
R_n
Initial
0 1 2 3 4 5 6 7 0
0 1 2 3 4 5 6 7
Initial
0
Request
Frame 0
Arrival
Arrival
ACK 1
Frame 0 delivered
1
Request
Frame 1
Lost
2
Request
Frame 2
Arrival
3
Request
Frame 3
NAK 1
Arrival
1
Arrival
Frame 1 (resend)
ACK 4
Arrival
Arrival
Frames 1, 2, 3 delivered