Data Link Layer

Data Link Layer

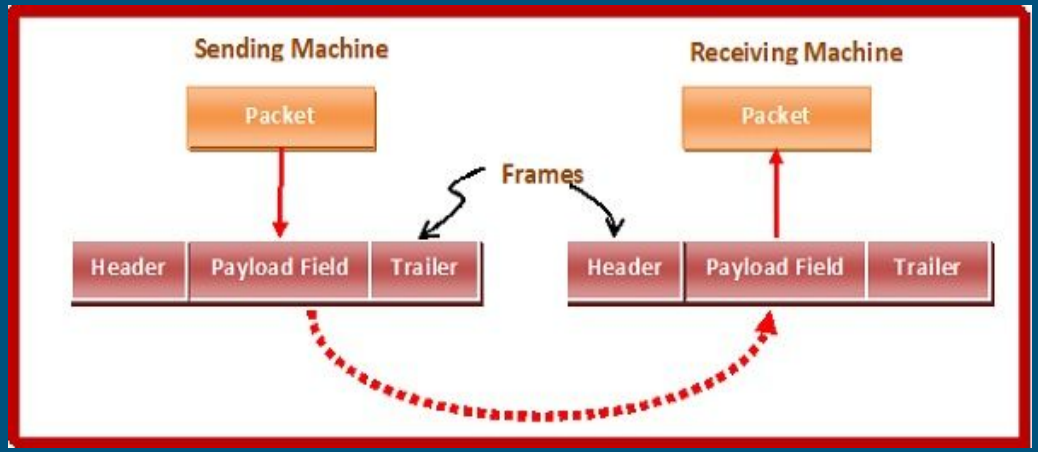
- **Design issues,**
  - **Services to network layer,**
  - **Framing- character count, character stuffing, bit stuffing, physical layer coding violation.**
  - **Error control, flow control,**
- **Elementary data link protocols-**
  - **Unrestricted simplex protocol**
  - **Simplex stop and wait protocol**
  - **Simplex protocol for a noisy channel**

Data Link Layer

- The Design principles of the data link layer deal with algorithms for achieving reliable, efficient communication of whole units of information called frames (rather than individual bits, as in the physical layer) between two adjacent machines.

- The data link layer uses the services of the physical layer to send and receive bits over communication channels.
- It has a number of functions, including:
  - Providing a well-defined service interface to the network layer.
  - Dealing with transmission errors.
  - Regulating the flow of data so that slow receivers are not swamped by fast senders

- The data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission
- Each frame contains
  - a frame header,
  - a payload field for holding the packet,
  - and a frame trailer



Data Link Layer Design Issues

a) Services Provided to the Network Layer

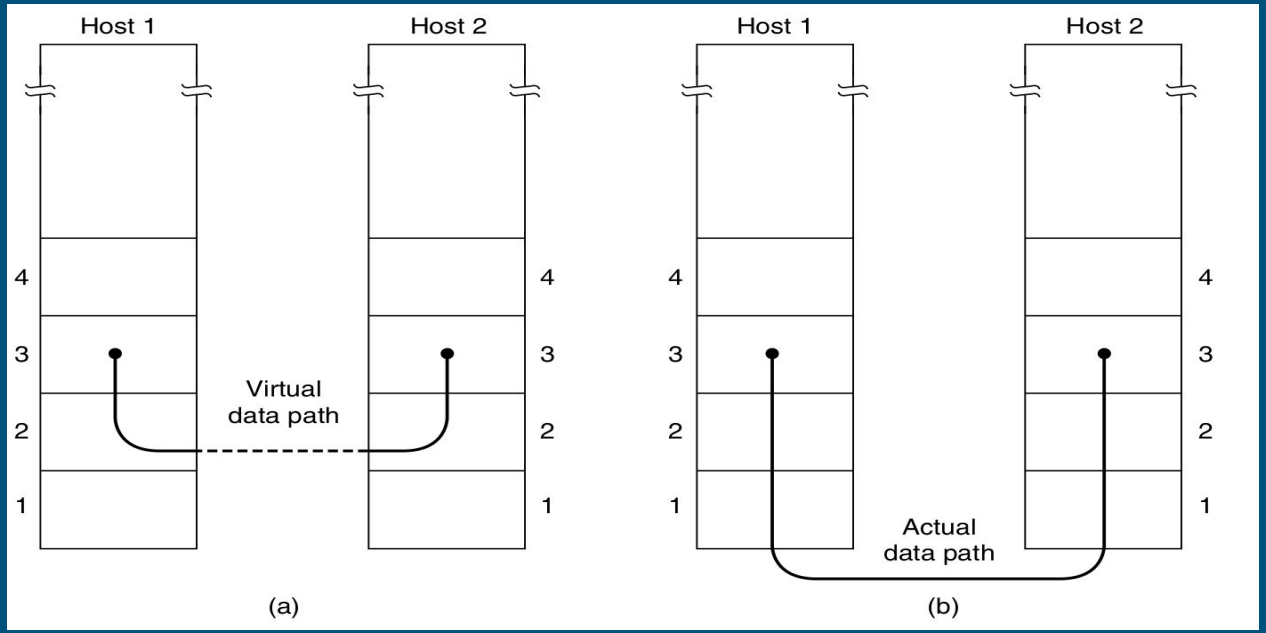
b) Framing

c) Error Control

d) Flow Control

Services Provided to the Network Layer

- The job of the data link layer is to **transmit** the bits to the destination machine as shown in Figure a
- The actual transmission follows the path of Figure b



- The data link layer can be designed to offer various services.
- The actual services that are offered vary from protocol to protocol.
- Three possibilities are
  - 1. Unacknowledged connectionless service.
  - 2. Acknowledged connectionless service.
  - 3. Acknowledged connection-oriented service.

- **Unacknowledged connectionless service**
  - **the source machine send independent frames to the destination machine**
  - **without having the destination machine acknowledge them**
- **If a frame is lost due to noise on the line, no attempt is made to detect the loss or recover from it in the data link layer**
- **This class of service is appropriate <u>when the error rate is very low</u>**

- **Acknowledged connectionless service**
  - **here are still no logical connections used,**
  - **but each frame sent is individually acknowledged.**
- **In this way, the sender knows whether a frame has arrived correctly or been lost.**
- **If it has not arrived within a specified time interval, it can be resent**

- **Connection-oriented service**
  - **the source and destination machines establish a connection before any data are transferred.**
- **Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received.**
- **Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order.**

- **Connection-oriented services go through three distinct phases**
  - **Connect**
  - **Transfer**
  - **Disconnect**

Framing

- ➔ **bit stream received by the data link layer is not guaranteed to be error-free**
- ➔ **It is up to the data link layer to detect and, if necessary, correct errors.**

- **Usual approach is for the data link layer**
  - **to break up the bit stream into discrete frames,**
  - **compute a short token called a checksum for each frame,**
  - **and include the checksum in the frame when it is transmitted**

- ➔ When a frame arrives at the destination, the checksum is recomputed.
- ➔ If the newly computed checksum is different from the one contained in the frame,
  - ◆ the data link layer knows that an error has occurred and takes steps to deal with it

Types of framing – There are two types of framing:

1. Fixed size – The frame is of fixed size, and there is no need to provide boundaries to the frame;

the length of the frame itself acts as a delimiter.

- Drawback: It suffers from internal fragmentation if the data size is less than the frame size
- Solution: Padding

❖ Variable Size

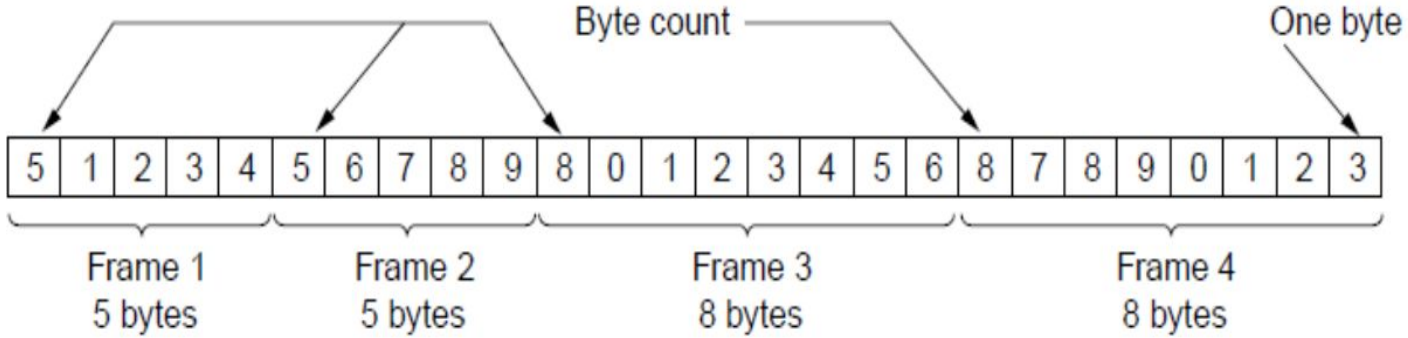
➢ **Breaking up the bit stream into frames—four methods:**

- **1. Byte count/Character count**
- **2. Flag bytes with byte stuffing/ Starting and ending characters, with character stuffing**
- **3. Flag bits with bit stuffing/ Starting and ending flags, with bit stuffing**
- **4. Physical layer coding violations.**

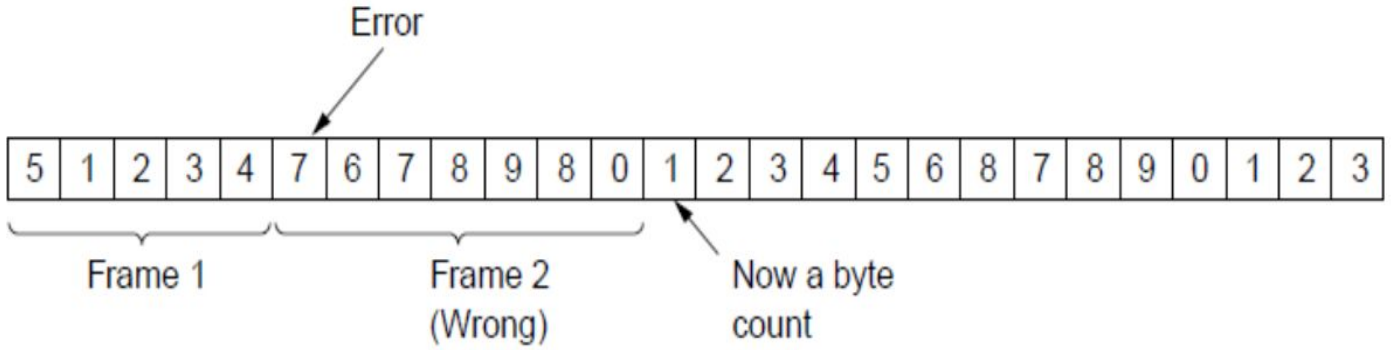
Byte Count method

- Frame begins with a count of the number of bytes in it
  - Simple but difficult to resynchronize after an error
  - The trouble with this algorithm is that the count can be distorted by a transmission error
  - Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many bytes to skip over to get to the start of the retransmission.
    - For this reason, the byte count method is rarely used by itself.

Expected case



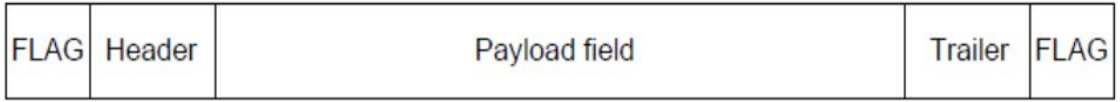
Error case



Byte stuffing method

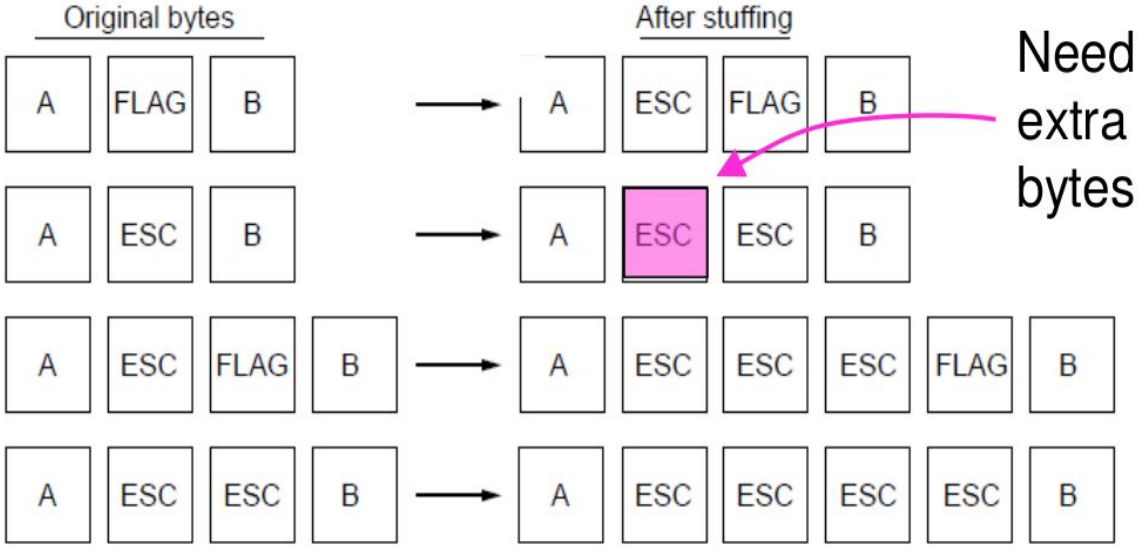
- This method gets around the problem of resynchronization after an error by having each frame start and end with special bytes.
  - Often the same byte, called a flag byte, is used as both the starting and ending delimiter.
- occurrences of flags in the data must be stuffed (escaped) with escape sequence
  - Longer, but easy to resynchronize after error

Frame format



(a)

Stuffing examples



Need to escape extra ESCAPE bytes too!

Example here is a simplification of PPP (point-to-point protocol)

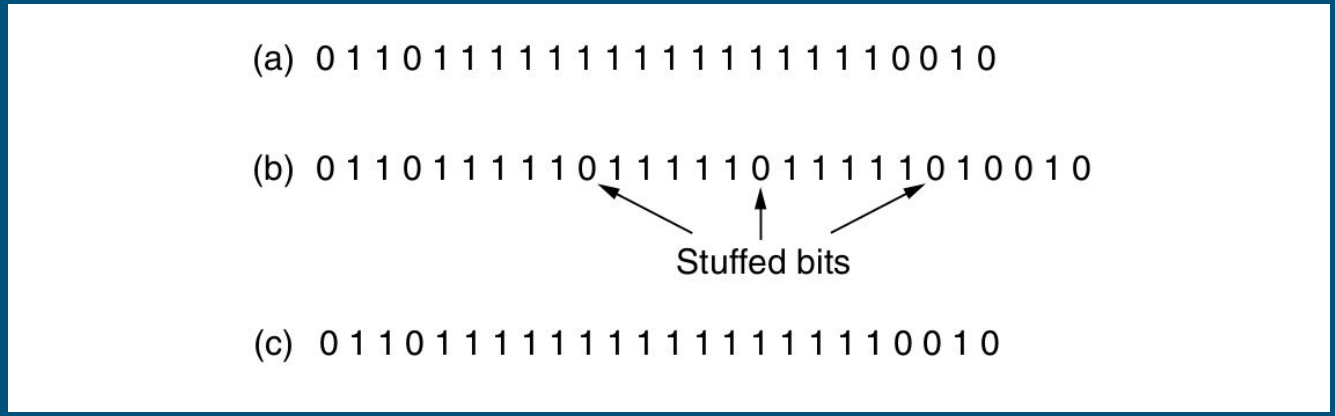
- If flag byte occurs in the data
  - insert a special escape byte (ESC) just before each "accidental" flag byte in the data
  - The data link layer on the receiving end removes the escape bytes before giving the data to the network layer.
- This technique is called byte stuffing.
  - The byte-stuffing scheme depicted here is a slight simplification of the one used in PPP (Point-to-Point Protocol)

Bit stuffing method

- Framing can be also be done at the bit level
- It was developed for the HDLC (High-level Data Link Control) protocol.
- Stuffing done at the bit level:
  - Frame flag has six consecutive 1s

Bit stuffing method

- Each frame begins and ends with a special bit pattern, 01111110 or 0x7E in hexadecimal. This pattern is a flag byte.
- Whenever the sender's data link layer encounters five consecutive 1s in the data,
  - it automatically stuffs a 0 bit into the outgoing bit stream
  - On transmit, after five 1s in the data, a 0 is added
  - On receive, a 0 after five 1s is deleted
- When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit

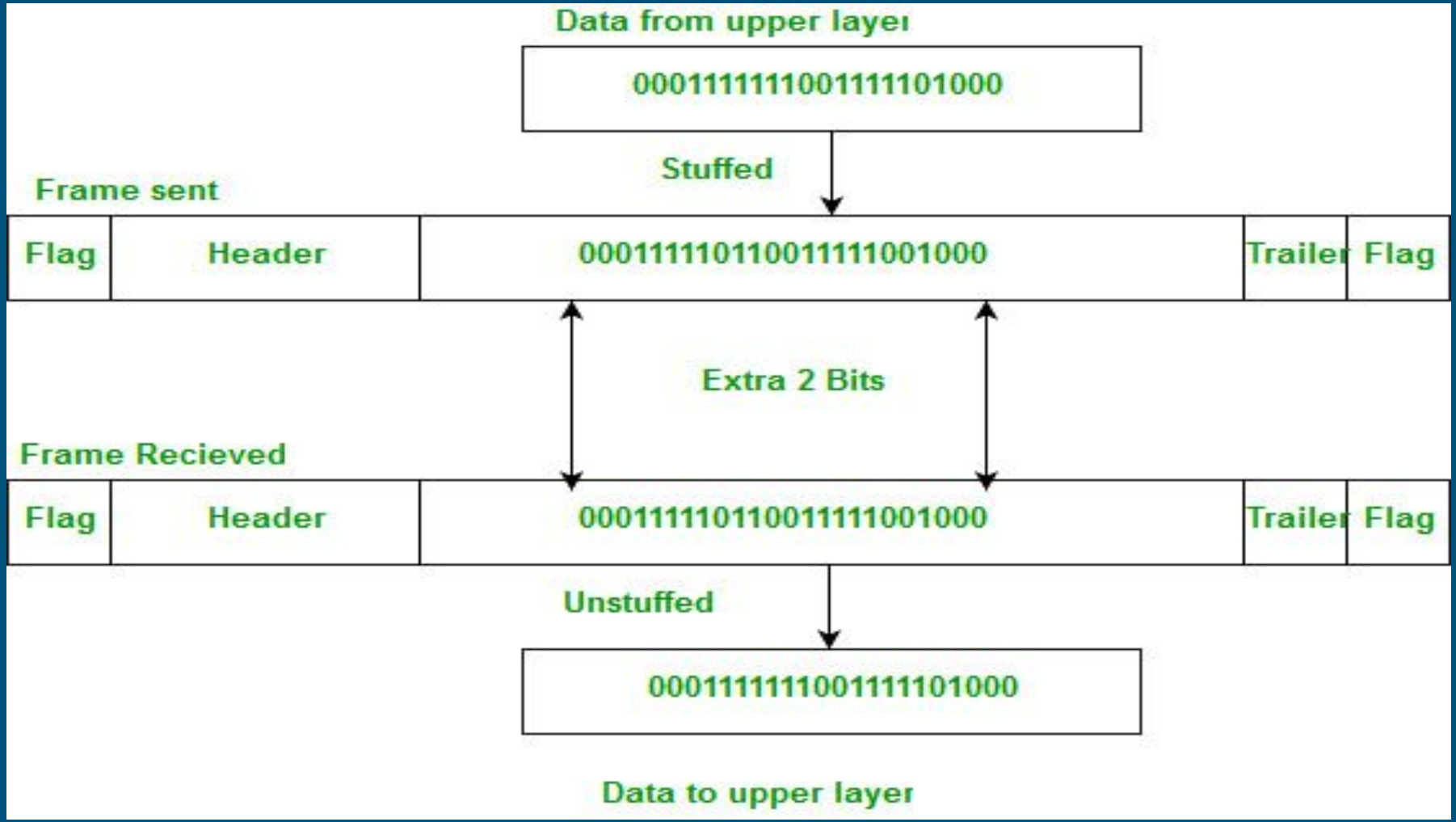


Bit stuffing

(a) The original data.

(b) The data as they appear on the line.

(c) The data as they are stored in the receiver's memory after destuffing.


Data from upper layer
00011111111001111101000
Stuffed
Frame sent
Flag
Header
000111110110011111001000
Trailer
Flag
Extra 2 Bits
Frame Recieved
Flag
Header
000111110110011111001000
Trailer
Flag
Unstuffed
00011111111001111101000
Data to upper layer

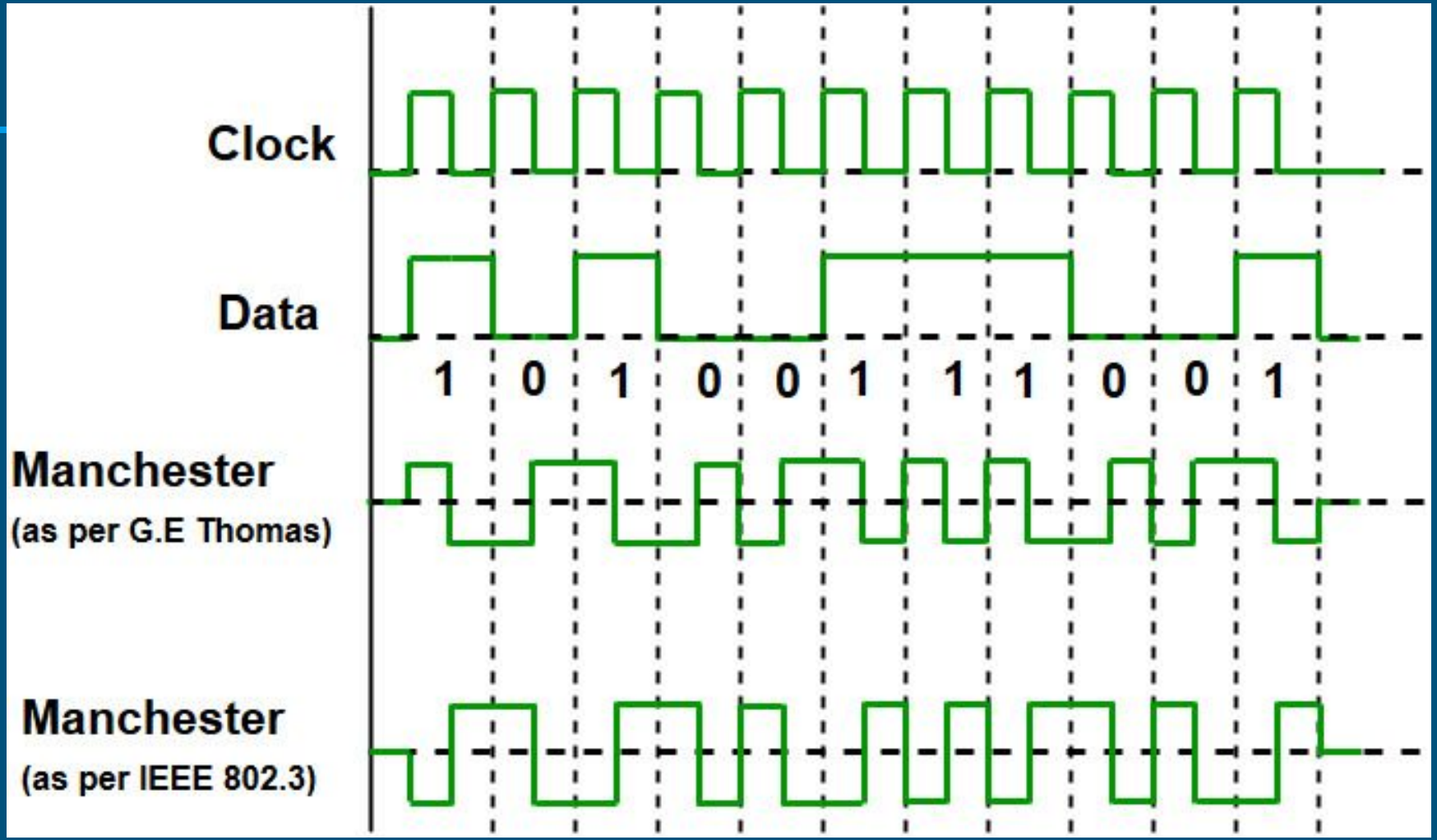
- **With both bit and byte stuffing, a side effect is that the length of a frame now depends on the contents of the data it carries.**
  - **For instance, if there are no flag bytes in the data, 100 bytes might be carried in a frame of roughly 100 bytes.**
  - **If, however, the data consists solely of flag bytes, each flag byte will be escaped and the frame will become roughly 200 bytes long.**
  - **With bit stuffing, the increase would be roughly 12.5% as 1 bit is added to every byte.**

Physical layer coding violations

- It is applicable to networks in which the encoding on the physical medium contains some redundancy.
- This redundancy means that some signals will not occur in regular data.
  - For example, in the 4B/5B line code,
    - 4 data bits are mapped to 5 signal bits to ensure sufficient bit transitions.
    - 2^4=16 combinations
    - 2^5=32 combinations
      - [32-16] (balance 16 combinations can be used as delimiters)

Physical layer coding violations

- Some LANs encode each bit of data by using two physical bits that Manchester coding uses.
- In such cases, normally,
  - a 1 bit is a high-low pair 10
  - a 0 bit is a low-high pair. 01
- The combinations of low-low and high-high
  - which are not used for data may be used for marking frame boundaries.


Clock
Data
1 0 1 0 0 1 1 1 0 0 1
Manchester
(as per G.E Thomas)
Manchester
(as per IEEE 802.3)

Physical layer coding violations

- Some reserved signals are used to indicate the start and end of frames.
- As they are reserved signals, it is easy to find the start and end of frames.
- Here we are using "coding violations" by putting reserved signals in original data.

- Many data link protocols use a combination of these methods for safety.
- A common pattern used for Ethernet and 802.11 is to have a frame begin with a well-defined pattern called a preamble
- This pattern has 72 bits , typical for 802.11
- The preamble is then followed by a length (i.e., count) field in the header that is used to locate the end of the frame.

Error Control

- **Error control is to make sure all frames are eventually delivered to the network layer at the destination and in the proper order.**
- **The usual way to ensure reliable delivery is to provide the sender with**
  - **some feedback**
  - **Unacknowledged connectionless service - sender just output frames without acknowledgement**
  - **For reliable, service - acknowledgement is important**

- **usual way to ensure reliable delivery**
  - **the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements**
  - **If the sender receives a positive acknowledgment about a frame, it knows the frame has arrived safely.**
  - **On the other hand, a negative acknowledgment means that something has gone wrong and the frame must be retransmitted**

- **Some of the hardware troubles**
  - **may cause a frame to vanish completely (in noise burst) - the receiver will not react**
  - **Or if the acknowledgement frame is lost - the sender will not know how to proceed**
- **It is dealt by introducing timers into the data link layer**

- When the sender transmits a frame, it generally also **starts a timer.**
- The timer is set to expire after an interval long enough for the frame to reach the destination and receive acknowledgement,
- Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out,
  - in which case the timer will be canceled.

- if either the frame or the acknowledgement is lost
- the timer will go off, alerting the sender to a potential problem.
- The obvious solution is to just retransmit the frame
  - By the time frames may be transmitted multiple times
  - receiver will accept the same frame two or more times
    - To prevent this from happening, it is generally necessary to assign sequence numbers to outgoing frames so that the receiver can distinguish retransmissions from originals.

Flow Control

- What to do with a sender wants to transmit frames faster than the receiver can accept them
- Two approaches are commonly used.
  - In the first one, **feedback-based flow control**,
    - the receiver sends back information to the sender giving it permission to send more data, or at least telling the sender how the receiver is doing.
  - In the second one, **rate-based flow control**,
    - the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.

Techniques of Flow Control in the Data Link Layer:

There are basically two types of techniques being developed to control the flow of data.

- Stop and Wait
- Sliding Window

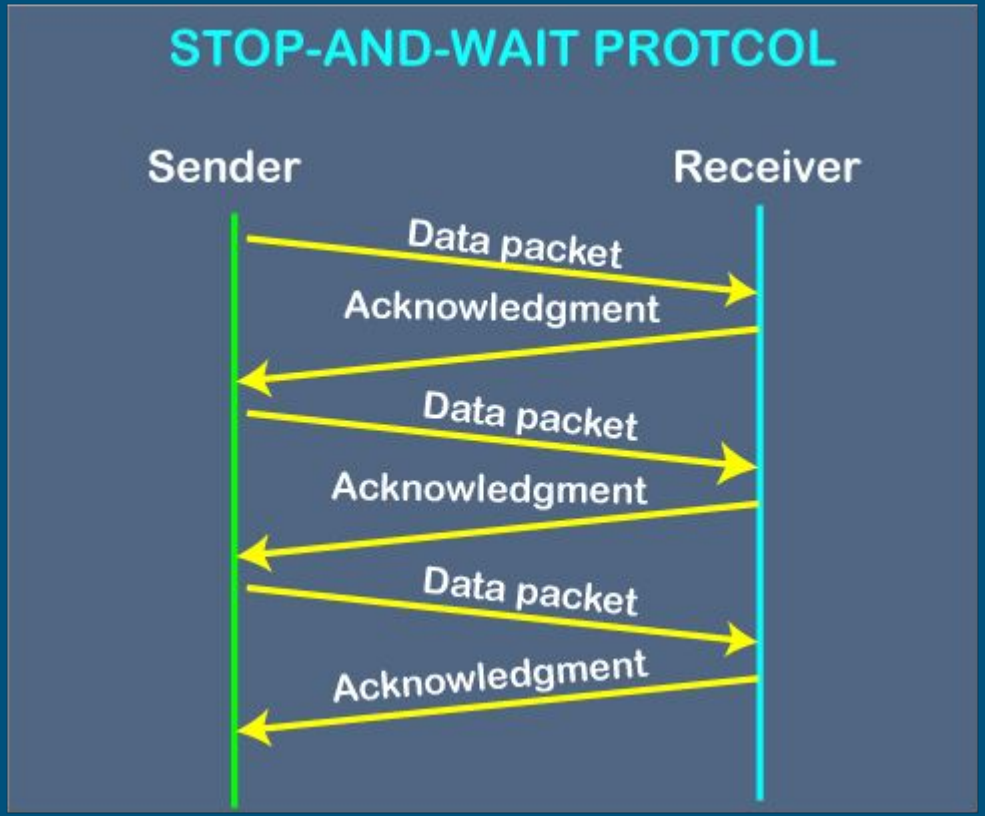
- Stop and Wait protocol and the Sliding Window protocol are the techniques to the solution of flow control handling.
- The main difference between the Stop-and-wait protocol and the Sliding window protocol is that
  - in the Stop-and-Wait Protocol, the sender sends one frame and waits for acknowledgment from the receiver
  - in the Sliding Window protocol, the sender sends more than one frame to the receiver and re-transmits the frame(s) which is/are damaged or suspected.

Stop-and-Wait Protocol

Stop-and-Wait is a simple protocol used for transmitting data between two devices over a communication channel. In this protocol, the sender sends a packet of data to the receiver and then waits for the receiver to acknowledge the packet before sending the next packet. The receiver sends an acknowledgment to the sender indicating that the packet has been received and is error-free.

Features

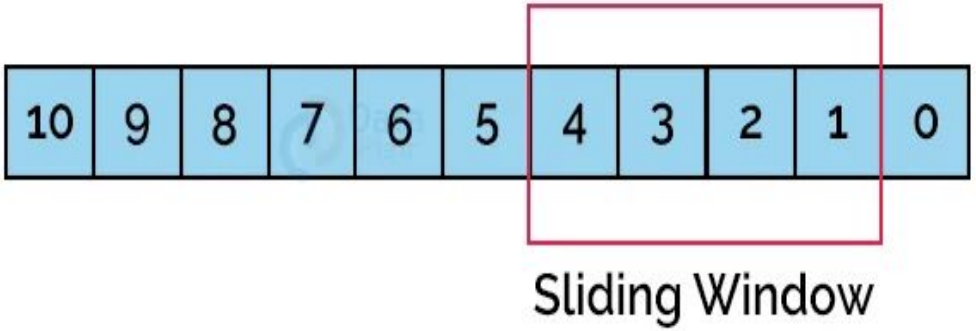
- The sender transmits one packet at a time and waits for an acknowledgment before sending the next packet.
- The receiver sends an acknowledgment for each packet received, indicating whether it is a duplicate or a new packet.
- It is a simple and easy-to-implement protocol.
- It has low efficiency compared to the sliding window protocol, as it requires a lot of time to wait for an acknowledgment for each packet.
- It is ideal for situations where the transmission rate is low or the network is reliable.


STOP-AND-WAIT PROTCOL
Sender
Receiver
Data packet
Acknowledgment
Data packet
Acknowledgment
Data packet
Acknowledgment

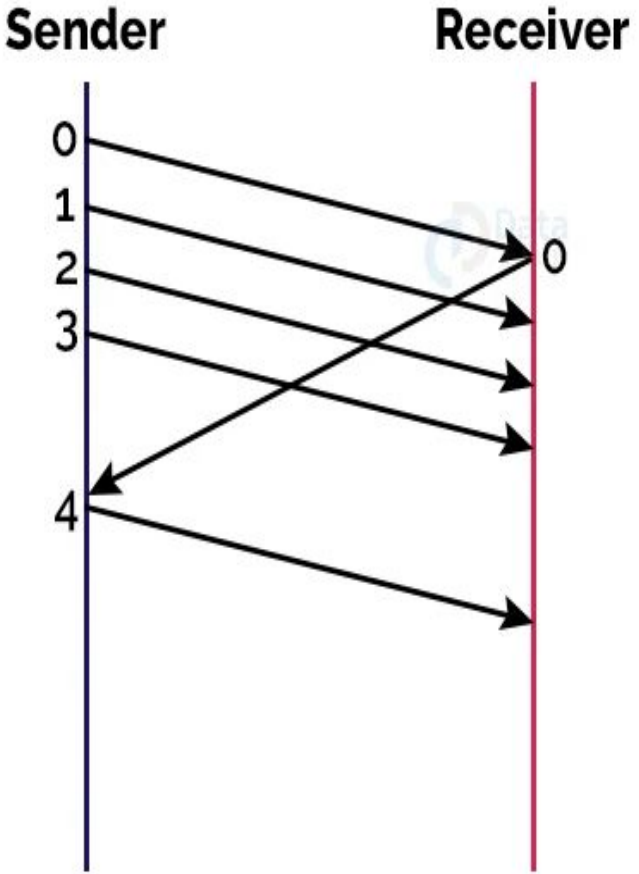
Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames.

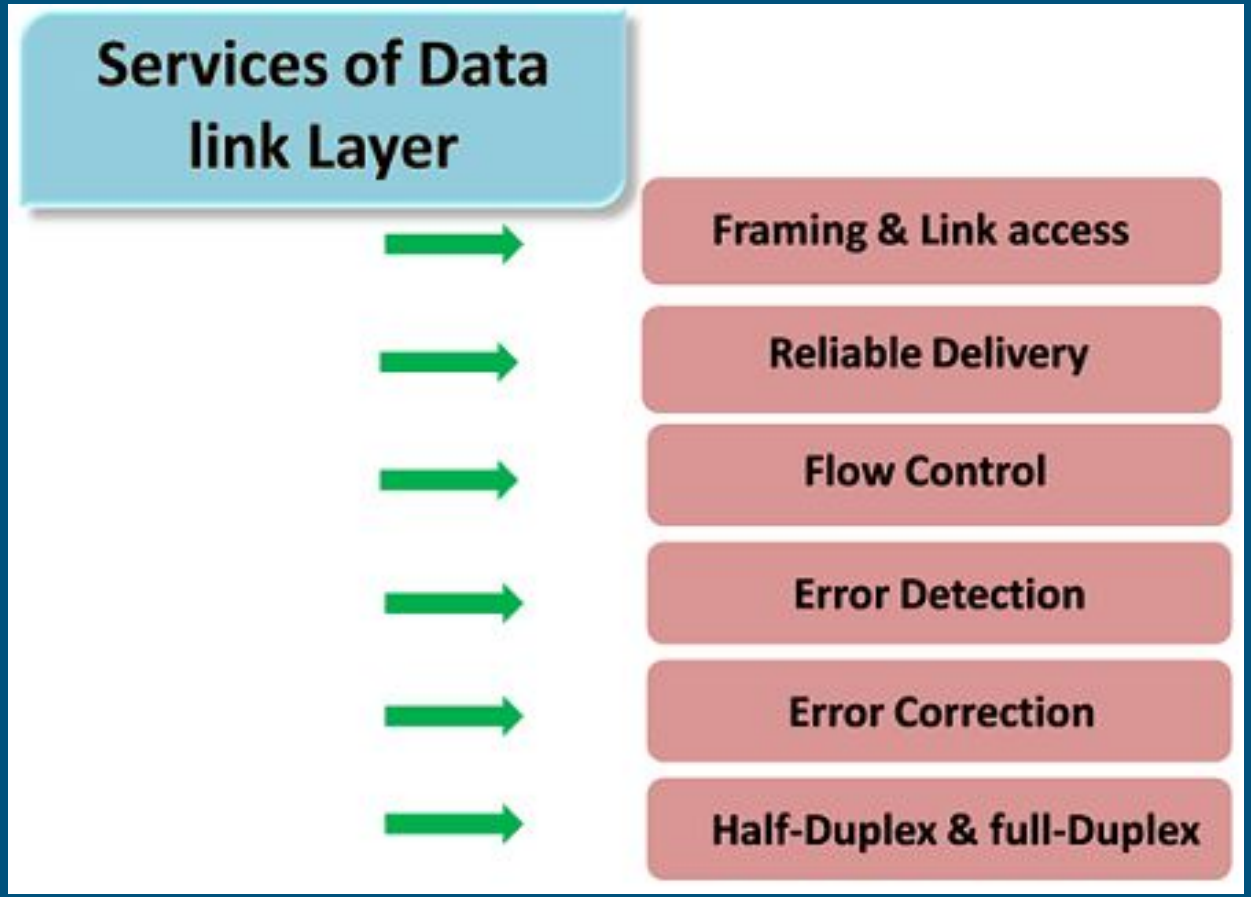
Multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver.

The term sliding window refers to the imaginary boxes to hold frames. Sliding window method is also known as windowing.



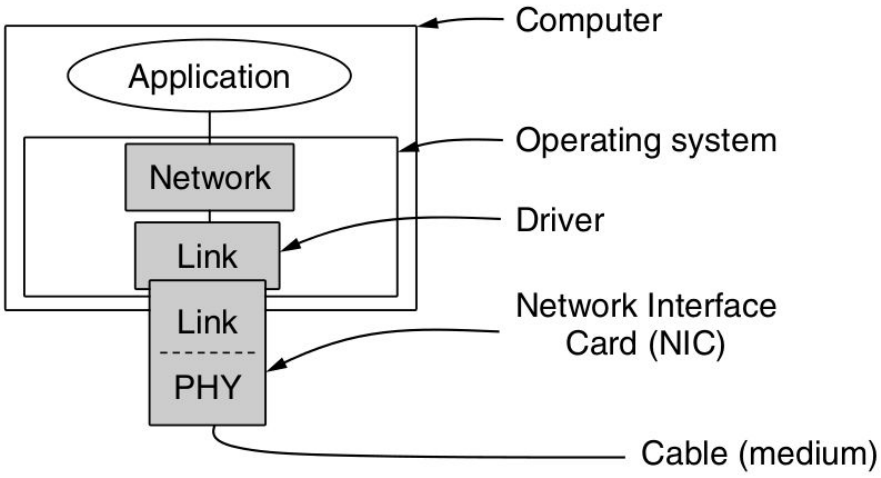
Window Size : 4




Services of Data link Layer
Framing & Link access
Reliable Delivery
Flow Control
Error Detection
Error Correction
Half-Duplex & full-Duplex

ELEMENTARY DATA LINK PROTOCOLS

- We assume that the physical layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth.
- A common implementation is shown in Fig



Implementation of the physical, data link, and network layers.

- When the data link layer accepts a packet, it encapsulates the packet in a frame by adding a data link header and trailer to it
- a frame consists of
  - an embedded packet,
  - some control information (in the header),
  - and a checksum (in the trailer)

header	Packet info	trailer

a frame consists of

- **Frame Header**: It contains the source and the destination addresses of the frame and the control bytes.
- **Payload field**: It contains the message to be delivered.
- **Trailer**: It contains the error detection and error correction bits. It is also called a Frame Check Sequence (FCS).
- **Flag**: Two flag at the two ends mark the beginning and the end of the frame.

- Fields of a Frame Header
  - A frame header contains the destination address, the source address and three control fields: kind, seq, ack and serving the following purposes:
    - kind: This field states whether the frame is a data frame or it is used for control functions like error and flow control or link management, etc.
    - seq: This contains the sequence number of the frame for rearrangement of out-of- sequence frames and sending acknowledgements by the receiver.

Piggybacking

- In two-way communication, when a data frame is received, the receiver waits and does not send the control frame (acknowledgement or ACK) back to the sender immediately.
- The receiver waits until its network layer passes in the next data packet. The delayed acknowledgement is then attached to this outgoing data frame.
- This technique of temporarily delaying the acknowledgement so that it can be hooked with next outgoing data frame is known as piggybacking.

Data link Protocol functions…….

Group	Library Function	Description
Network layer	from_network_layer(&packet) to_network_layer(&packet) enable_network_layer() disable_network_layer()	Take a packet from network layer to send Deliver a received packet to network layer Let network cause “ready” events Prevent network “ready” events
Physical layer	from_physical_layer(&frame) to_physical_layer(&frame)	Get an incoming frame from physical layer Pass an outgoing frame to physical layer
Events & timers	wait_for_event(&event) start_timer(seq_nr) stop_timer(seq_nr) start_ack_timer() stop_ack_timer()	Wait for a packet / frame / timer event Start a countdown timer running Stop a countdown timer from running Start the ACK countdown timer Stop the ACK countdown timer

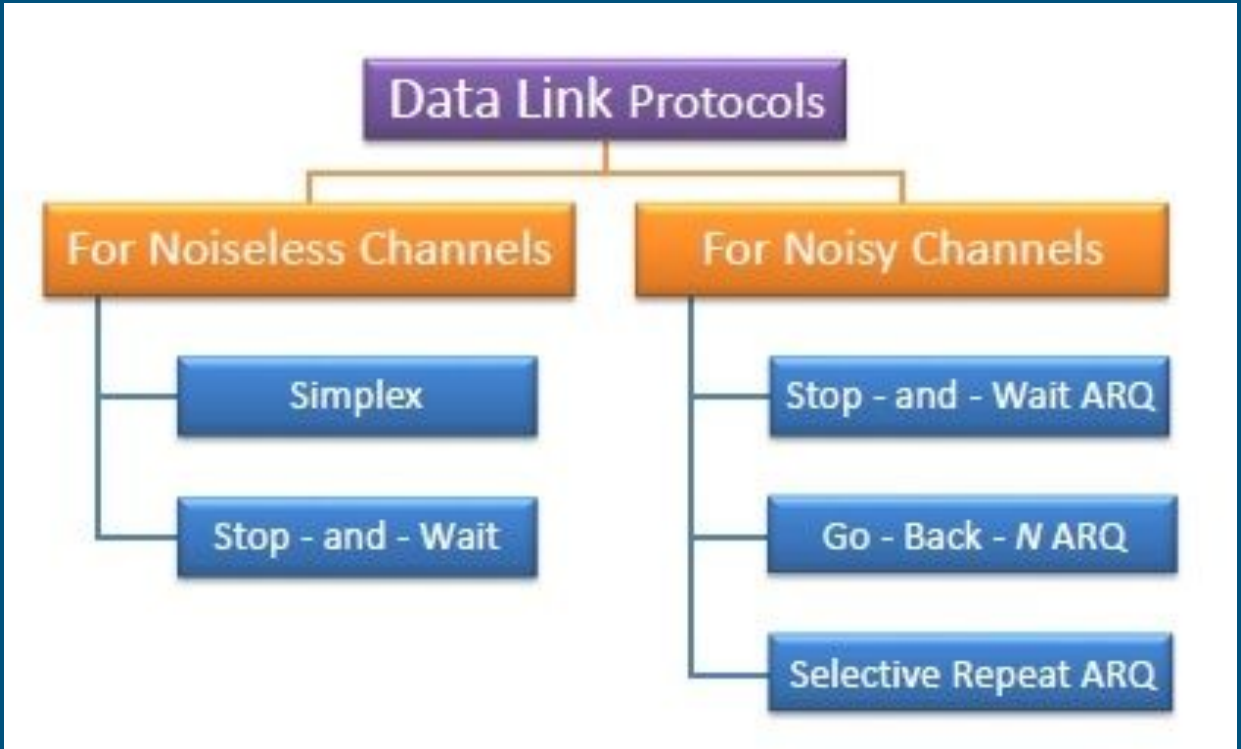
Function definitions are located in the file protocol.h.

- Some of the declarations, common to many of the protocols
  - A **boolean** is an enumerated type—true and false.
  - A **seq nr** is a small integer used to number the frames.
    - These sequence numbers are from 0 to MAX_SEQ

- **A packet is the unit of information exchanged between the network layer with MAX_PKT bytes (which would be of variable length).**
  - **Though it contains MAX PKT bytes, more realistically it would be of variable length.**

- There exist suitable library procedures such as
  - to_physical_layer()
    - to send a frame, i.e., frame to PL
  - from_physical_layer()
    - to receive a frame, i.e., a frame from PL
- These procedures
  - compute and append or check the checksum.
- Also functions for events/timers/network layer

ELEMENTARY DATA LINK PROTOCOLS

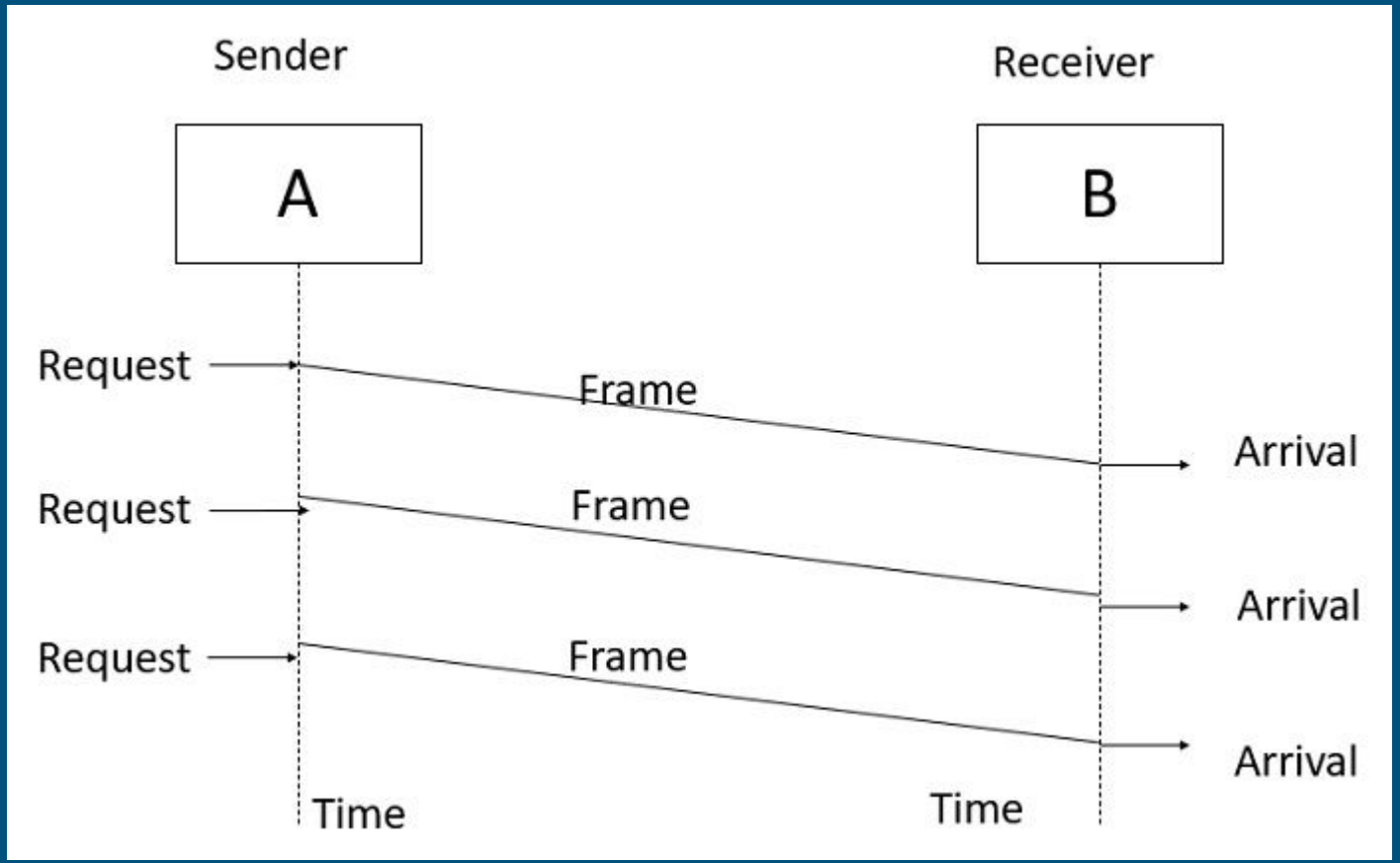


Elementary Data Link protocols are classified into three categories

a. Protocol 1 – Unrestricted simplex protocol
b. Protocol 2 – Simplex stop and wait protocol
c. Protocol 3 – Simplex protocol for noisy channels.

- Protocol 1 –
  - Unrestricted simplex protocol

○ Protocol 1 – Unrestricted simplex protocol



Utopian Simplex Protocol/ Unrestricted Simplex Protocol

- Simple protocol
- because it does not worry about the possibility of anything going wrong
- Assumes no errors
- and receiver as fast as sender
- Considers one-way data transfer

```
void sender1(void)
{
  frame s;
  packet buffer;

  while (true) {
       from_network_layer(&buffer);
       s.info = buffer;
       to_physical_layer(&s);
  }
}
```

Sender loops by continuously sending the frames

```
void receiver1(void)
{
  frame r;
  event_type event;

  while (true) {
       wait_for_event(&event);
       from_physical_layer(&r);
       to_network_layer(&r.info);
  }
}
```

Receiver loops by consuming/receiving the frames

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

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* send it on its way */
  }


}
```

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

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

- This protocol is unrealistic because
  - it does not handle either
    - flow control
    - or error correction.

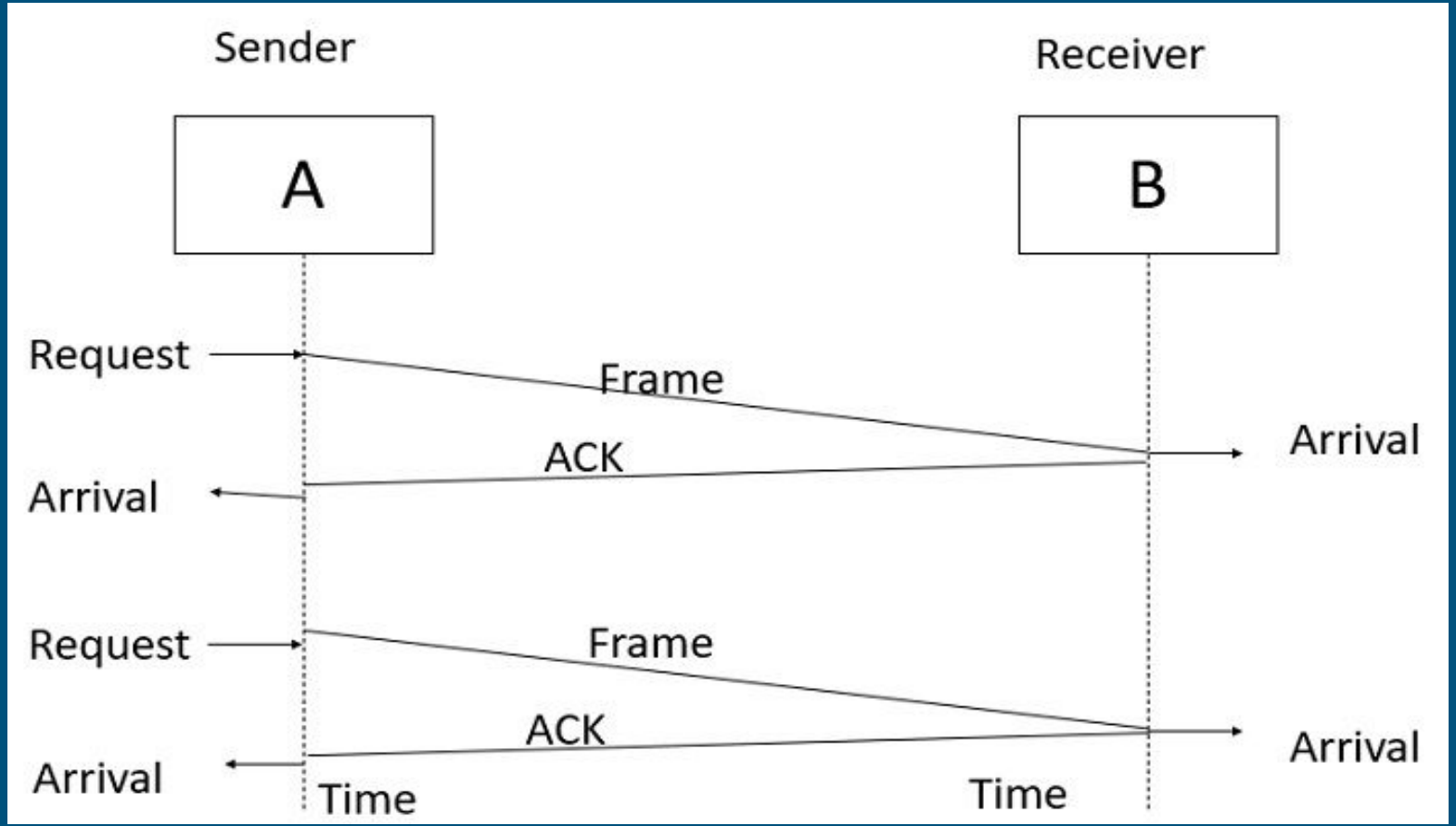
- Protocol 2 –
  - Simplex stop and wait protocol
  - For an error-free channel

- **In unrestricted simplex protocol,**
  - **the sender flooding the receiver happens, which is a problem**
- **Which can be overcome in the**
  - **Simplex stop and wait protocol**

- **Solution**
  - **is to build the receiver to be powerful enough to process a continuous stream of back-to-back frames (or, equivalently, define the DLL to be slow enough that the receiver can keep up).**
    - **For which it must have sufficient buffering.**
    - **This is a worst-case solution.**
    - **It requires dedicated H/W & can be wasteful of resources**

- **A more general solution**
  - **to this problem is to have the receiver provide feedback to the sender.**
  - **After having passed a packet to its network layer, the receiver sends a small dummy frame (Control Frame) back to the sender, which, in effect, gives the sender permission to transmit the next frame.**

○ Protocol 2 – Simplex stop and wait protocol



- After sending a frame, the sender waits for a time period until a small dummy (i.e., acknowledgement) frame arrives.
- This delay is a simple example of a flow control protocol.
- Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait protocols

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

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

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

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

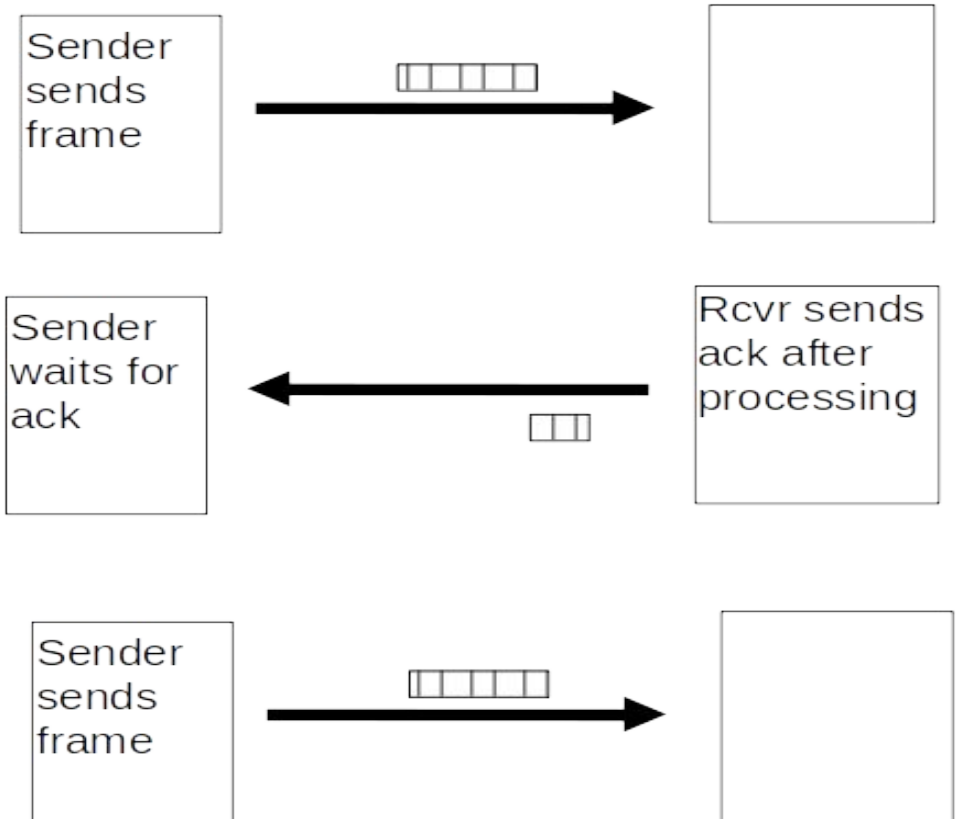
A simplex stop-and-wait protocol.

Protocol summary

- Receiver returns a dummy frame (ack) when ready
- Only one frame is sent at a time— called stop-and-wait
- Therefore added flow control!
- The communication channel is still assumed to be error free.
- The sender never transmits a new frame until the old one's ack has been fetched by the physical layer
- The protocols in which the sender sends a frame and then waits for an acknowledgment before proceeding are called **Stop and Wait.**

Simplex stop-and-wait protocol

Assumption . Channel never looses data



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

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

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

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

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

- Protocol 3 –
  - Simplex protocol for noisy channels.

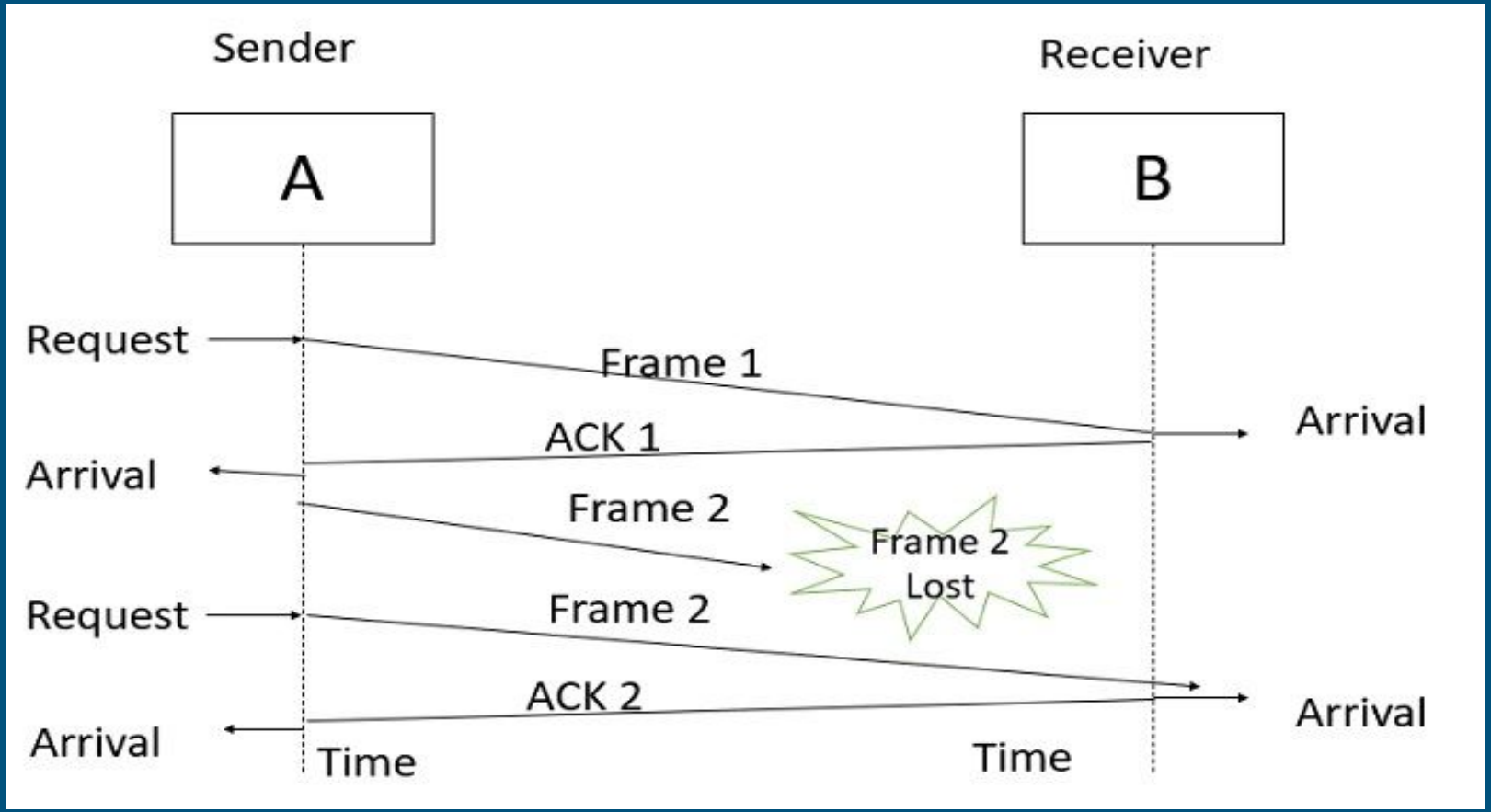
Simplex Protocol for a Noisy Channel

Assumptions:

1) Data transfer is only in one direction.
2) Separate sender and receiver.
3) Finite processing capacity and speed at the receiver
4) Since it is a noisy channel , errors in the data frames or acknowledgement frames are expected.
5) Every frame has an unique sequence number.
6) After a frame has been transmitted, timer is started for a finite time. Before the timer expires, if the acknowledgement is not received, the frame gets retransmitted.
7) When the acknowledgement gets corrupted or the sent data frame gets damaged, how long the sender should wait to transmit the next frame is infinite.

- **The normal situation of a communication channel is that it makes errors.**
- **Frames may be either damaged or lost completely.**
- **Assumption.**
  - *** Data frame can be lost**
  - *** Ack can be lost**

- **Protocol 3 – Simplex protocol for noisy channels.**



- **Lost data frames:**
  - This means that sender will never get an ack.
    - Sender can implement timer for each frame sent. If timer expires retransmit.
- **Lost ACK frames**
  - But what if the data frame arrived but the ack was lost?
  - sender will retransmit frame that receiver has already received.

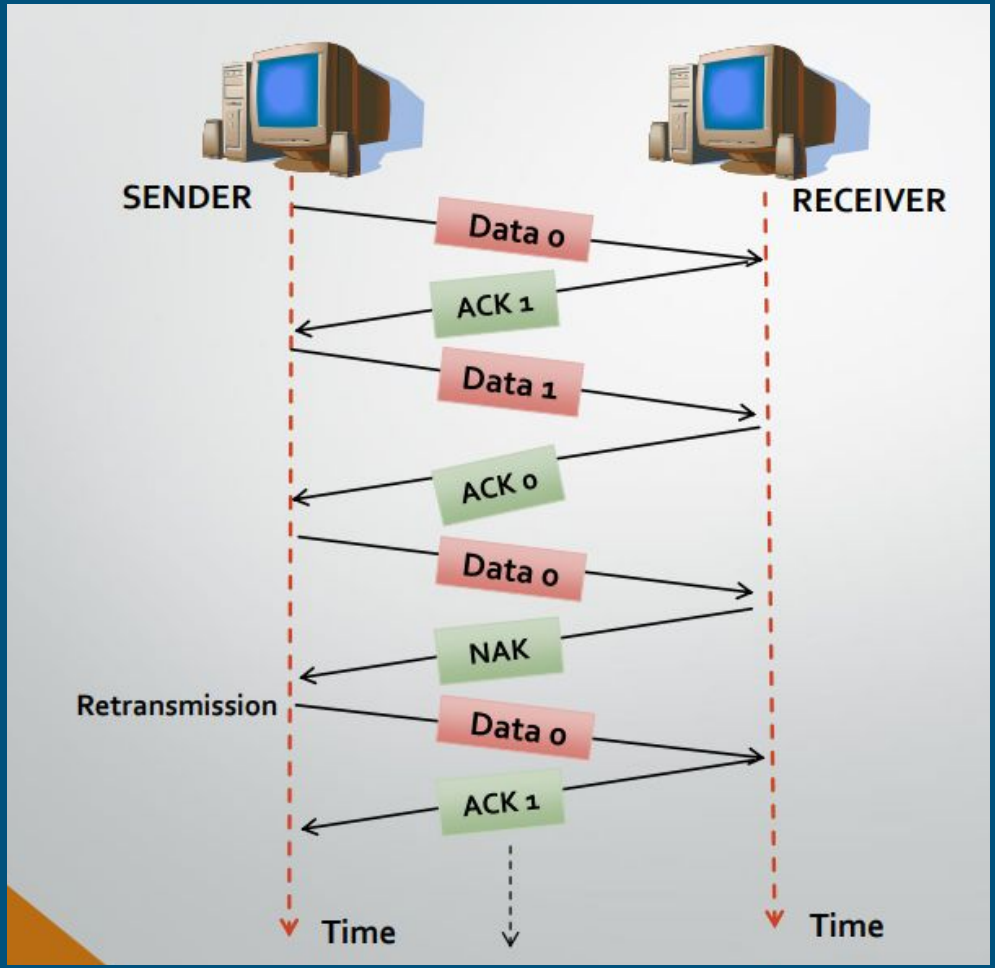
- **Result: Duplicate frame at sender.**
  - **So the receiver needs some way of distinguishing duplicates.**
  - **Answer: Use sequence numbers.**

- **What should be the field size of these sequence numbers ?**
  - **Since we are still using a stop-and-wait type protocol,**
    - **sender will never transmit frame m+1 unless it gets an ack for m.**
  - **So the receiver needs to only distinguish between successive frames.**
  - **A 1 bit sequence number (0,1) is enough.**
  - **Sender includes consecutive seq numbers in frame, receiver sends ack with seq number.**

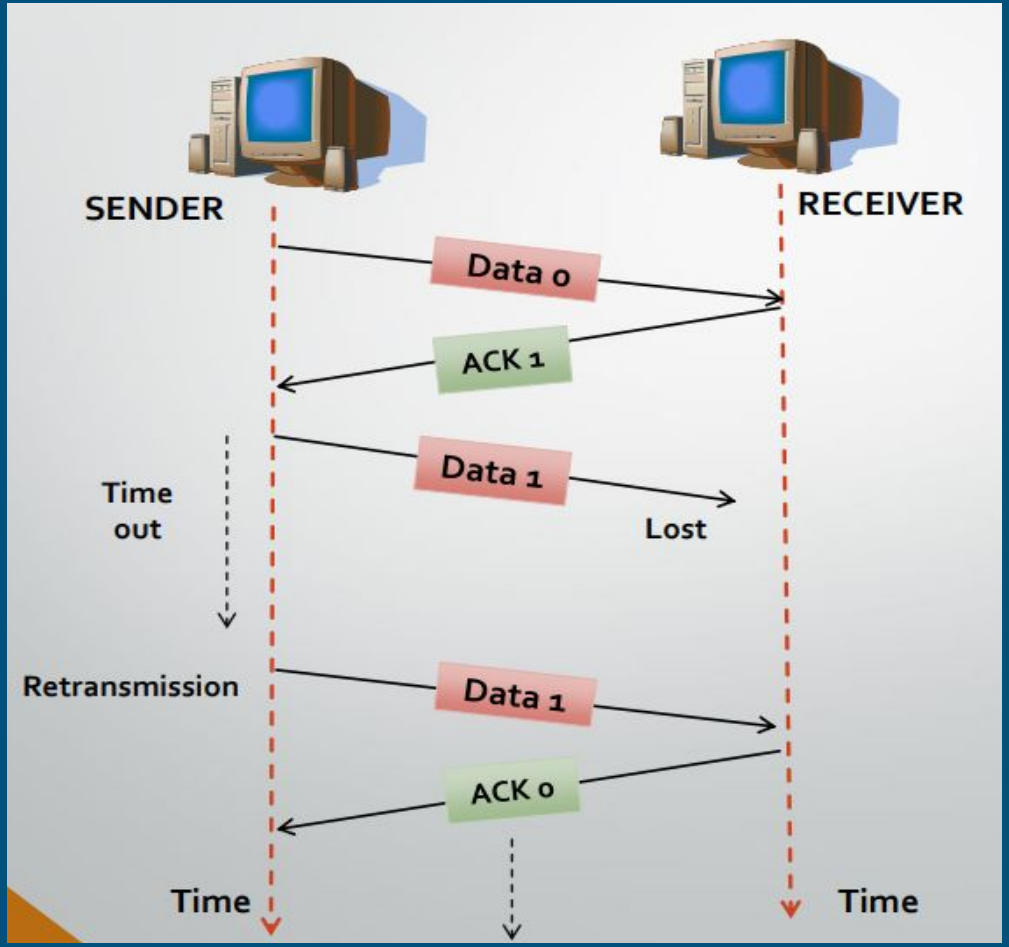
- **The main difference between this protocol for simple stop-and-wait is that the**
  - **receiver knows which seq num to expect and sender knows which seq num to send.**
- **This technique is called *Automatic Repeat Request* (ARQ) or *PAR (Positive Ack with re-transmission)***

- **ARQ (Automatic Repeat reQuest) adds error control**
  - **Receiver acks frames that are correctly delivered -**
  - **Also known as Positive Acknowledgement with Retransmission (PAR)**
- **Sender sets timer and resends frame if no ack**
  - **For correctness, frames and acks must be numbered**
  - **Else receiver can’t tell retransmission (due to lost ack or early timer) from new frame**
- **For stop-and-wait, 2 numbers (1 bit) are sufficient**

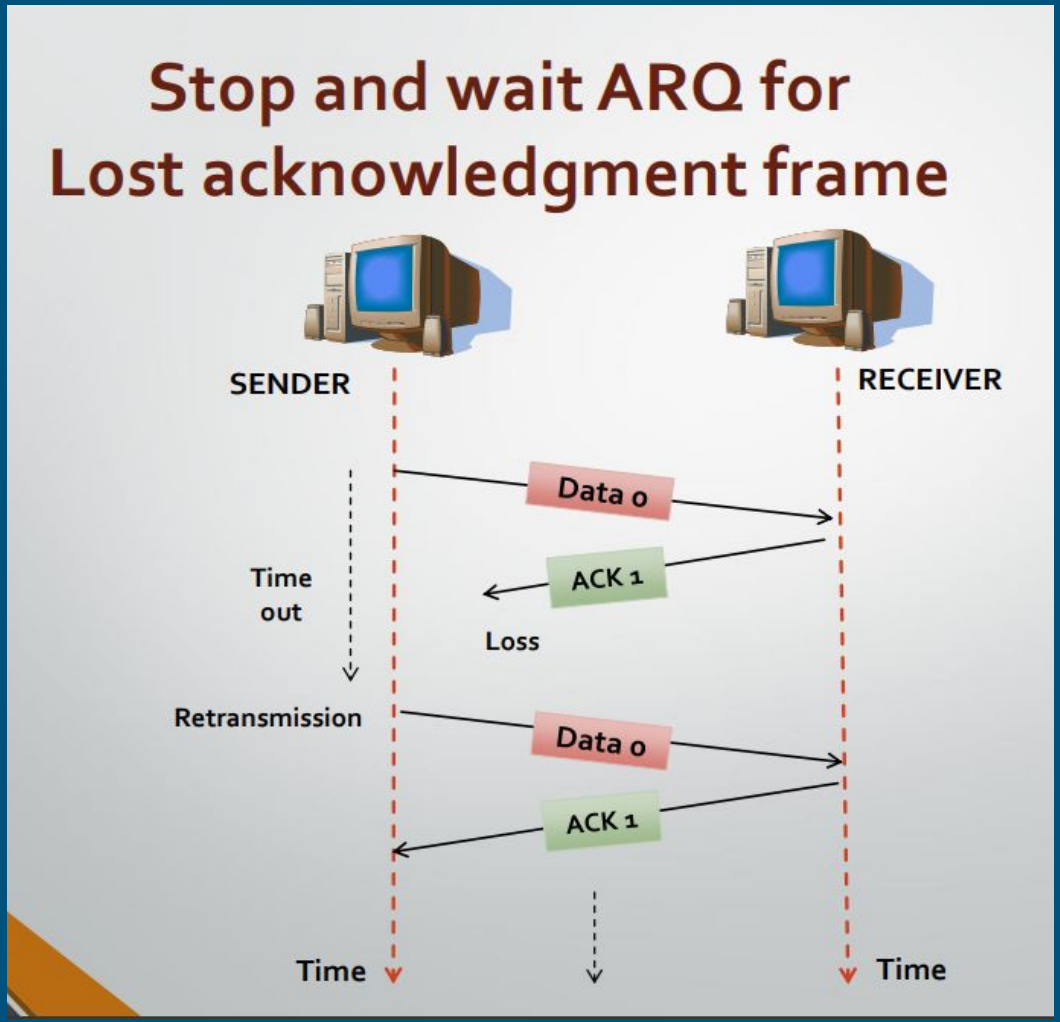
Stop and Wait ARQ for Damaged Data frames



Stop and wait ARQ for Lost data frames



Stop and wait ARQ for Lost acknowledgment frame



- **Sender:**
  - **Send frame with seq number N.**
  - **Wait (with timer) for ack with seq number N.**
  - **If timer expires: retransmit frame with seq number N**
  - **If ack arrives : N = mod(N+1,2)**

Sender loop (p3):

```
void sender3(void) {
  seq_nr next_frame_to_send;
  frame s;
  packet buffer;
  event_type event;

  next_frame_to_send = 0;
  from_network_layer(&buffer);
  while (true) {
      s.info = buffer;
      s.seq = next_frame_to_send;
      to_physical_layer(&s);
      start_timer(s.seq);
      wait_for_event(&event);
      if (event == frame_arrival) {
          from_physical_layer(&s);
          if (s.ack == next_frame_to_send) {
              stop_timer(s.ack);
              from_network_layer(&buffer);
              inc(next_frame_to_send);
          }
      }
  }
}
```

Send frame (or retransmission)

Set timer for retransmission

Wait for ack or timeout

If a good ack then set up for the next frame to send (else the old frame will be retransmitted)

- **Receiver:**
  - **Expect frame N**
  - **If N arrives:**
    - **deliver to network layer, send ack for N, N = mod(N+1,2)**
  - **If any other seq num: send previous ack (mod(N-1,2))**

Receiver loop (p3):

```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);
       if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                 to_network_layer(&r.info);
                 inc(frame_expected);
            }
            s.ack = 1 – frame_expected;
            to_physical_layer(&s);
       }
  }
}
```

Wait for a frame

If it's new then take it and advance expected frame

Ack current frame