
# CSC 524

Computer Networks

Dr. Esam A. Alwagait

Lecture 4

18-19/2/2013

# Agenda

1. Introduction
2. Data Link Layer Design Issues
3. Error Detection and Correction
4. Data Layer Protocols
5. Sliding Window Protocols
6. Summary & Discussion

# Introduction

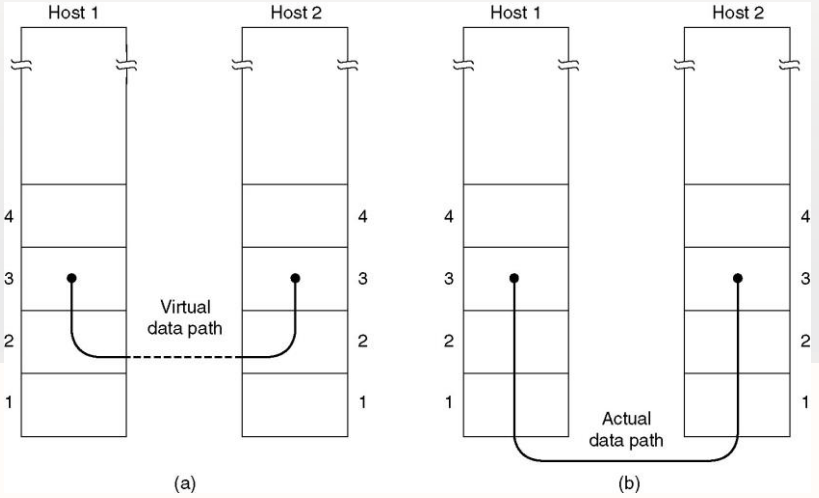

- Last week we discussed Physical Layer
- How do you send data from Network Layer to Physical layer ?
- How do you detect errors ?
  - Or Correct them ?
- Examples of Protocols for sender and receivers.
- And more !

# Agenda

1. Introduction
2. Data Link Layer Design Issues
3. Error Detection and Correction
4. Data Layer Protocols
5. Sliding Window Protocols
6. Summary & Discussion

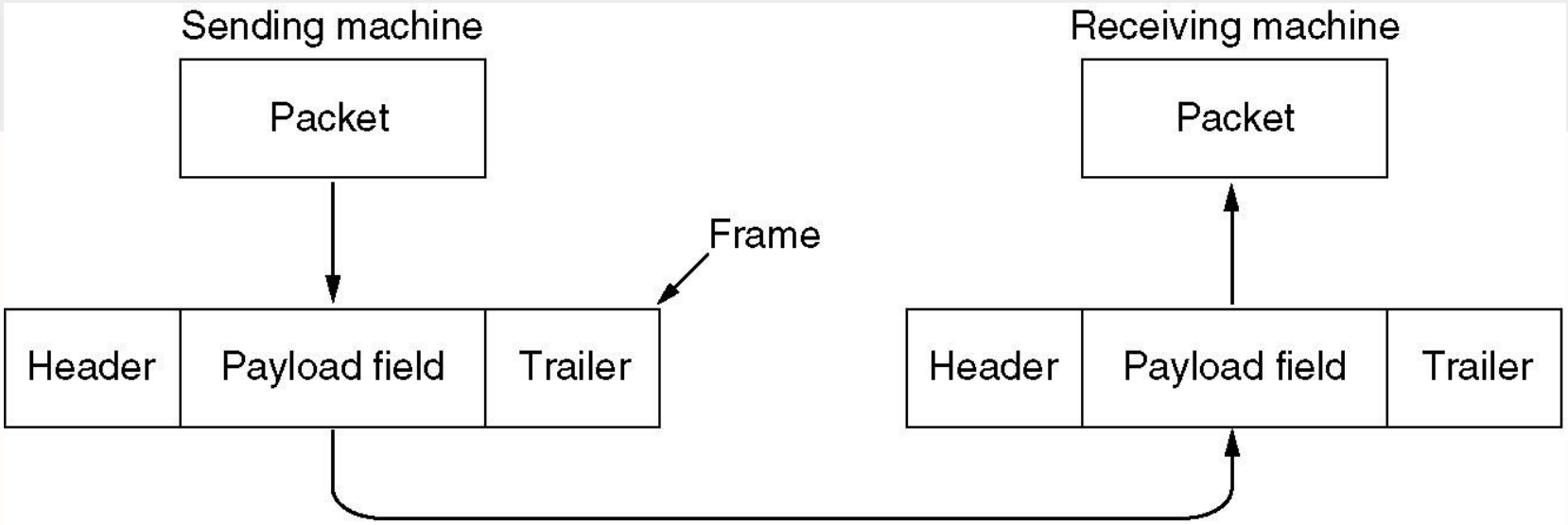
# Design Issues


- Functions of Data Link Layer:
  - 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.



# Design Issues (Cont'd)


- Data Link Layer (DL) Takes **packets** from network layer and transmits them in **Frames**
- Each frame has a **header** and a **Trailer**



# Design Issues (Cont'd)

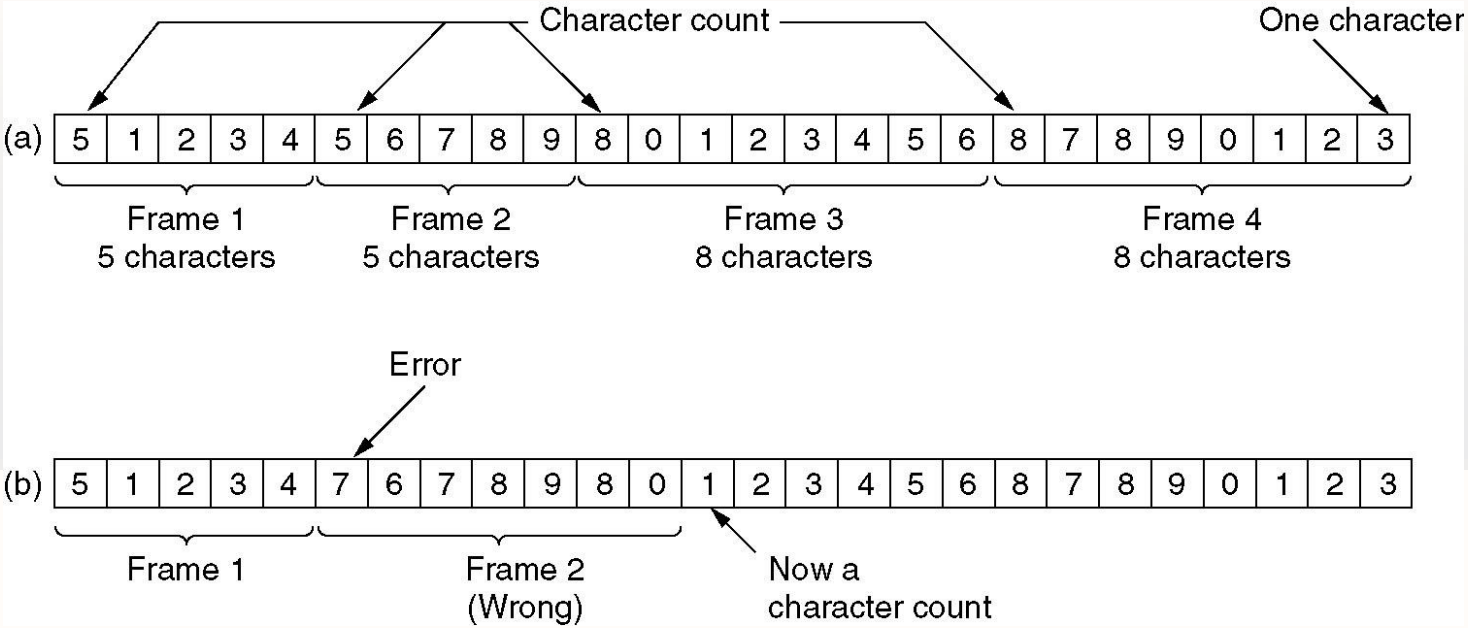

- **Notice:** many of the issues we will discuss can be found in other layers too. regardless, It is the concept that matters rather than the place where the functionality is implemented.
- Three services that are commonly provided are:
  - Unacknowledged connectionless service.
  - Acknowledged connectionless service.
  - Acknowledged connection-oriented service.

# Design Issues (Cont'd)


- Unacknowledged connectionless
  - No ack
  - No Error detection/correction
  - Assumption of a reliable channel
- acknowledged connectionless
  - Ack is used to determain reliable sending
- Connection-oriented acknowledged
  - Several frames are sent within a "channel"

# Design Issues (Cont'd)

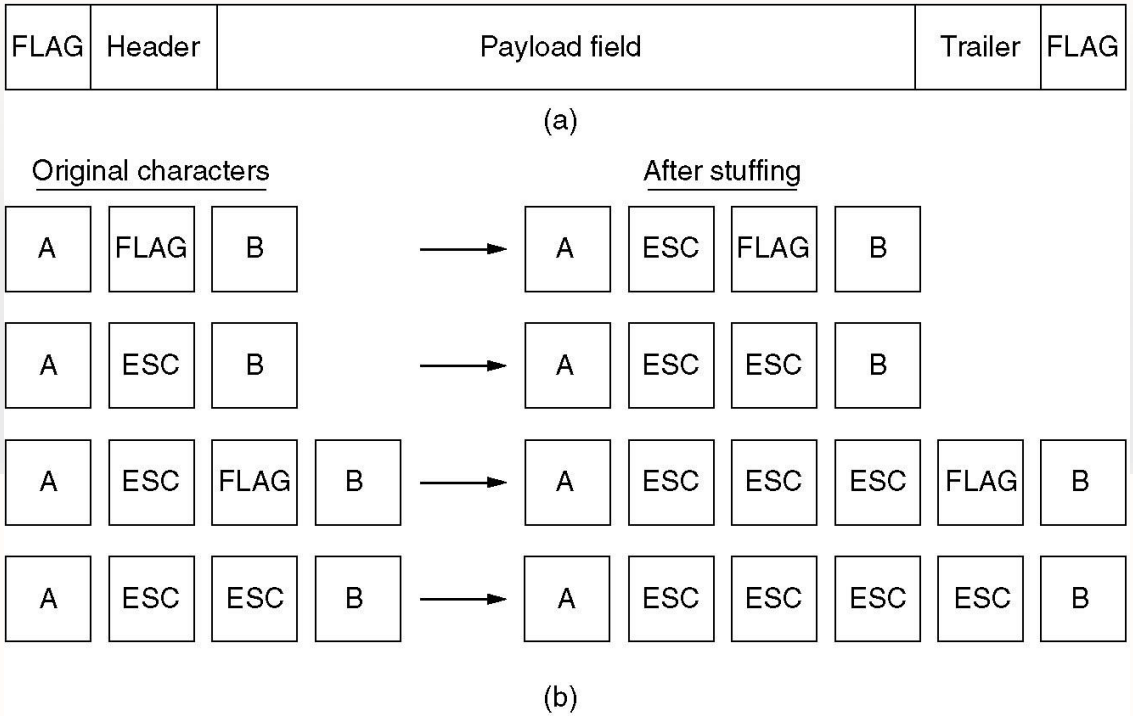

- Framing - Character Counting!




A character stream. (a) Without errors. (b) With one error.

# Design Issues (Cont'd)


- Framing - Flags

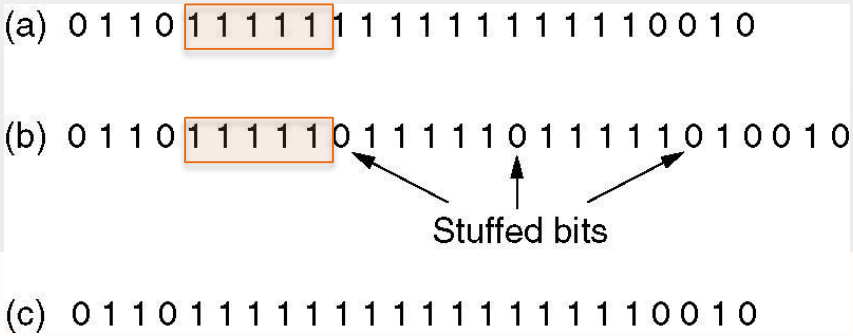



(a) A frame delimited by flag bytes.

(b) Four examples of byte sequences before and after stuffing.

# Design Issues (Cont'd)


- Byte stuffing assumes 8bit characters
  - UNICODE is 16 bit
  - Bit stuffing (0111110)




Bit stuffing

(a) The original data.

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

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

# Design Issues (Cont'd)


- Error  Control
  - Detection
  - Correction
- Flow Control
  - What if the sender is too Fast !
    - Feedback-based Flow control
    - Rate-based flow control
  - We will discuss them in future lectures

# Agenda

1. Introduction
2. Data Link Layer Design Issues
3. Error Detection and Correction
4. Data Layer Protocols
5. Sliding Window Protocols
6. Summary & Discussion

# Error Control


- Error control deals with problems that could occur in the physical layer!
- Must include **redundant** data !
  - How much ? Depends on the choice
- A choice has to be made whether to correct or just detect !
  - Detect requires re-transmission
  - Correction introduces extra cost
- Wireless is usually noisy, LAN is not !

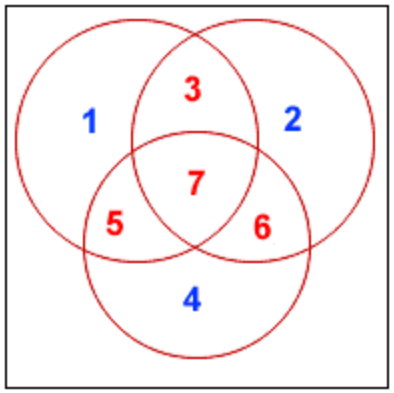
# Error Control


- Error Correction
  - Hamming Distance
    - Codeword is Message (m) + Redundant (r)
    - if two **codewords** are a Hamming distance **d** apart, it will require **d** single-bit errors to convert one into the other.
    - i.e you could reverse the error
    - Introducing Parity bit(s) (even/odd)
    - Single even parity bit
      - E.g. 1011010 becomes 1011010**0**
      - Odd is 1011010**1**

# Error Control


- Parity bits are in positions of the 2(n)
- E.g. 3 parity bits

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| D | D | D | P | D | P | P | 7-BIT CODEWORD |
| D | – | D | – | D | – | P | (EVEN PARITY) |
| D | D | – | – | D | P | – | (EVEN PARITY) |
| D | D | D | P | – | – | – | (EVEN PARITY) |

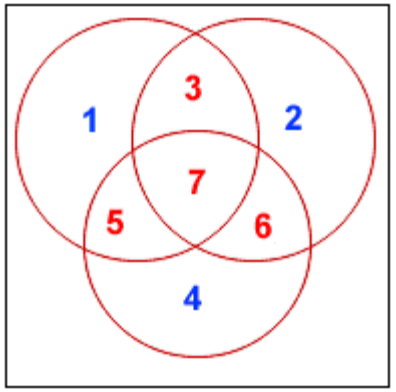



| 7 | 6 | 5 | 4 | 3 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 7-BIT CODEWORD |
| 1 | – | 0 | – | 1 | – | 0 | (EVEN PARITY) |
| 1 | 1 | – | – | 1 | 1 | – | (EVEN PARITY) |
| 1 | 1 | 0 | 0 | – | – | – | (EVEN PARITY) |

# Error Control


```
        transmitted message                          received message
        1 1 0 0 1 1 0          ------------>         1 1 1 0 1 1 0
BIT:    7 6 5 4 3 2 1                        BIT:    7 6 5 4 3 2 1
```

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 7-BIT CODEWORD |
| 1 | – | 1 | – | 1 | – | 0 | (EVEN PARITY) NOT! 1 |
| 1 | 1 | – | – | 1 | 1 | – | (EVEN PARITY) OK! 0 |
| 1 | 1 | 1 | 0 | – | – | – | (EVEN PARITY) NOT! 1 |




- 101 = 5 .. Bit 5 is wrong ! Reverse it
- Same principle for larger number of bits
- http://www.ee.unb.ca/tervo/ee4253/hamming.shtml

# Error Control


- Error detection
  - Error correction sometimes introduces an extra overhead that is not justified !
  - Bandwidth is consumed to send error correction data more than the data itself
  - The % of r/m !
  - If the error probability is not high, it is easier to retransmit

# Error Control


- How do you detect an error ?
  - Checksum !
  - Polynomial code → CRC (Cyclic Redundancy Check)
- Sender and receiver must agree on a Generator Function G(x)
- Checksums are used in other applications like storing passwords

# Agenda

1. Introduction
2. Data Link Layer Design Issues
3. Error Detection and Correction
4. Data Layer Protocols
5. Sliding Window Protocols
6. Summary & Discussion

# Data Layer Protocol


```
#define MAX_PKT 1024                                  /* determines packet size in bytes */

typedef enum {false, true} boolean;                   /* boolean type */
typedef unsigned int seq_nr;                          /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;             /* frame_kind definition */

typedef struct {                                      /* frames are transported in this layer */
  frame_kind kind;                                    /* what kind of a frame is it? */
  seq_nr seq;                                         /* sequence number */
  seq_nr ack;                                         /* acknowledgement number */
  packet info;                                        /* the network layer packet */
} frame;
```

# Data Layer Protocol (Cont'd)


```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

# Data Layer Protocol (Cont'd)


```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free,
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame  arrival} event  type;
#include "protocol.h"

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 */
 }                                  /  * Tomorrow, and tomorrow, and tomorrow,
                                      Creeps in this petty pace from day to day
                                      To the last syllable of recorded time
                                          - Macbeth, V, v */
}
```

```
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 */
 }
}
```

# Data Layer Protocol (Cont'd)


```
/* 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 */
  }
}
```

# Data Layer Protocol (Cont'd)


```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
 seq_nr next_frame_to_send;                         /* seq number of next outgoing frame */
 frame s;                                           /* scratch variable */
 packet buffer;                                     /* buffer for an outbound packet */
 event_type event;

 next_frame_to_send = 0;                            /* initialize outbound sequence numbers */
 from_network_layer(&buffer);                       /* fetch first packet */
 while (true) {
      s.info = buffer;                              /* construct a frame for transmission */
      s.seq = next_frame_to_send;                   /* insert sequence number in frame */
      to_physical_layer(&s);                        /* send it on its way */
      start_timer(s.seq);                           /* if answer takes too long, time out */
      wait_for_event(&event);                       /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
            from_physical_layer(&s);                /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                  stop_timer(s.ack);                /* turn the timer off */
                  from_network_layer(&buffer);      /* get the next one to send */
                  inc(next_frame_to_send);          /* invert next_frame_to_send */
            }
      }
 }
}
```

# Data Layer Protocol (Cont'd)


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

 frame_expected = 0;
 while (true) {
      wait_for_event(&event);                  /* possibilities: frame_arrival, cksum_err */
      if (event == frame_arrival) {            /* a valid frame has arrived. */
            from_physical_layer(&r);           /* go get the newly arrived frame */
            if (r.seq == frame  expected) {    /* this is what we have been waiting for. */
                  to_network_layer(&r.info);   /* pass the data to the network layer */
                  inc(frame_expected);         /* next time expect the other sequence nr */
            }
            s.ack = 1 – frame_expected;        /* tell which frame is being acked */
            to_physical_layer(&s);             /* send acknowledgement */
      }
 }
}
```

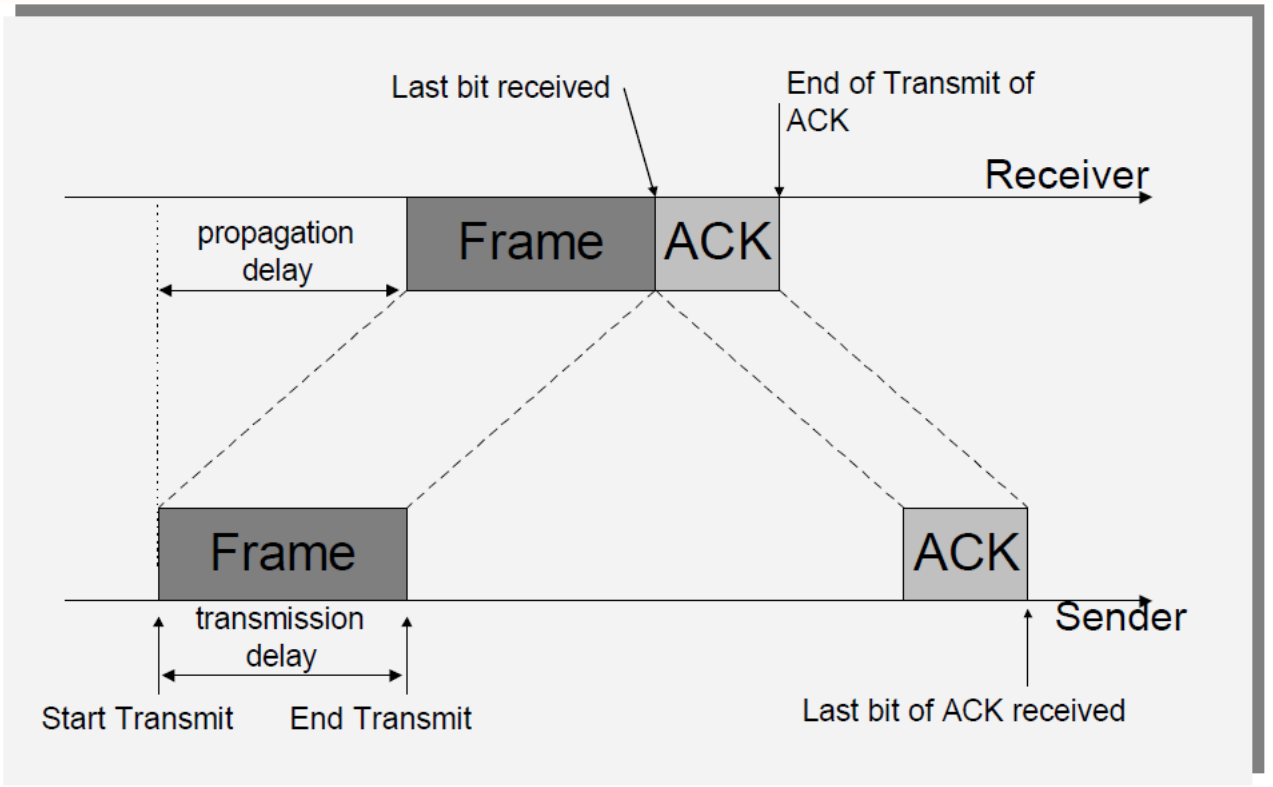
# Sliding window protocol


- Stop-and-wait wastes bandwidth !
- Simplest form of flow control
- In Stop-and-Wait flow control, the receiver indicates its readiness to receive data for each frame
- **Operations:**
  - *1.* **Sender:** Transmit a single frame
  - *2.* **Receiver:** Transmit acknowledgment (ACK)
  - *3.* Goto 1.

# Sliding window protocol(Cont'd)


- Transmission delay = time to send the data
  - E.g. 1000 bit over 1Mbps = 1msec
- Propagation delay = time from a bit is sent till it is received
  - Depends on medium and distance
  - E.g. 1000 km over guided media (200km/sec) is 5 msec

# Sliding window protocol(Cont'd)



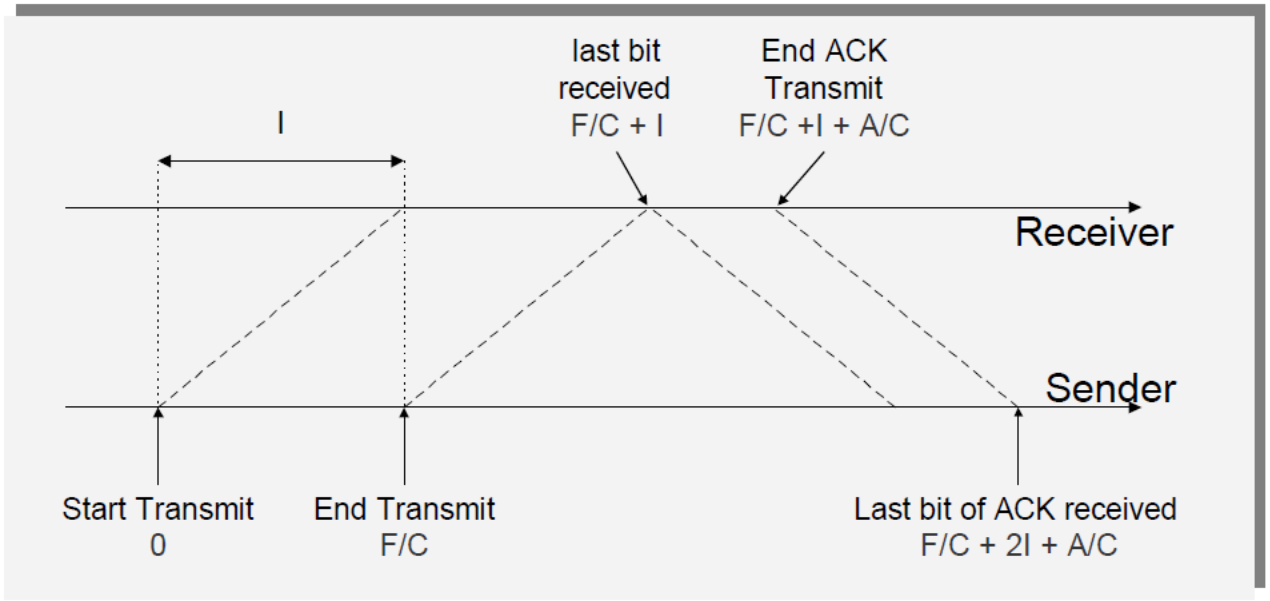
# Sliding window protocol(Cont'd)


## *Analysis of Stop-and-Wait*

- **Notation**:

| | |
|---|---|
| C | = Channel capacity in bps |
| I | = Propagation delay |
| H | = Number of bits in a frame header |
| D | = Number of data bits in a frame |
| F | = Total length of a frame (F= D+H) |
| A | = Total length of an ACK frame |
| F/C | = Transmission delay for a frame |

# Sliding window protocol(Cont'd)

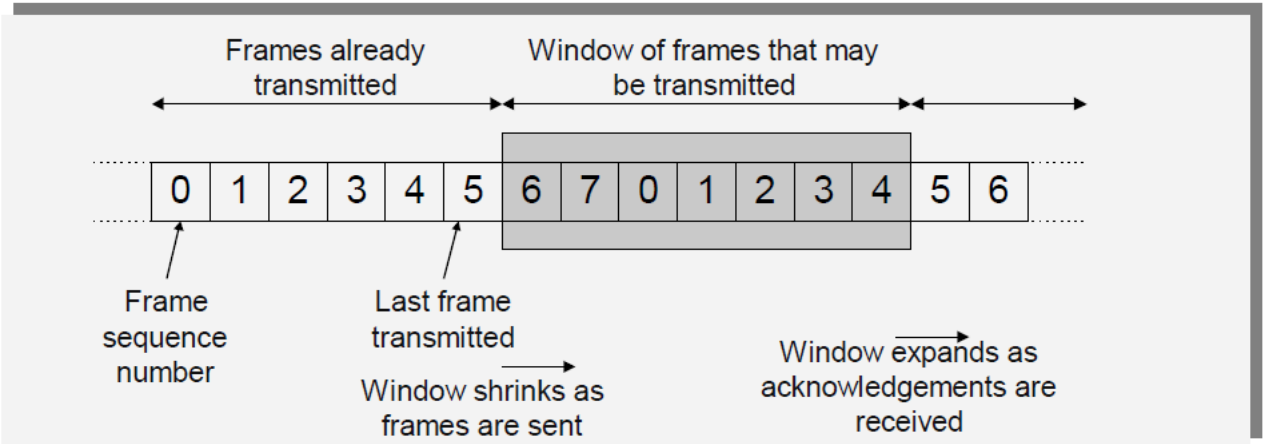


# Sliding window protocol(Cont'd)


- Sliding Window Protocol
  - Multiple frames transmitted at a time
  - Sending and receiver both have "windows"
  - Each frame has a sequence number
  - Window have lower+higher bounds
  - When receiving a frame receiver increases lower bound of the window and sends ack to the sender
  - Timeout ? Resend
  - Received something out of the window.. Drop it

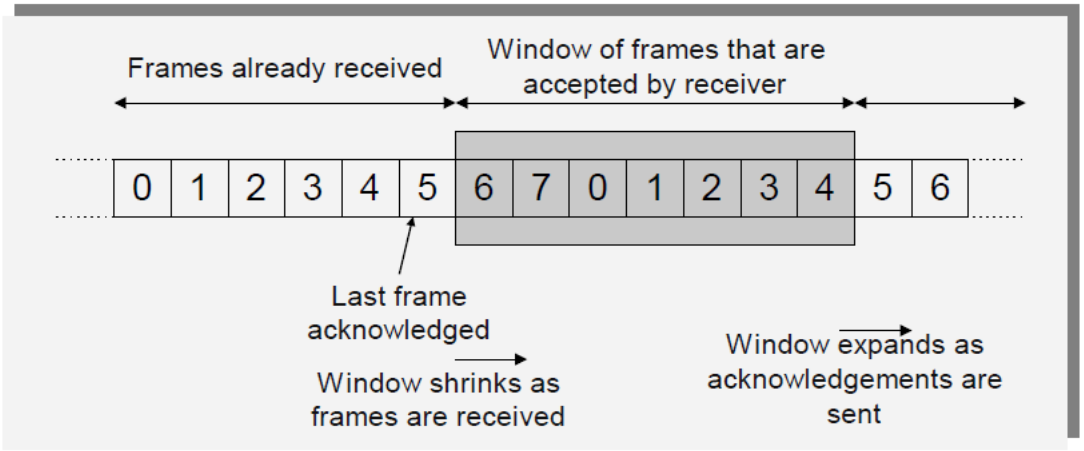
# Sliding window protocol(Cont'd)


- **Sending Window:**
  - At any instant, the sender is permitted to send frames with sequence numbers in a certain range
  - The range of sequence numbers is called the *sending window*



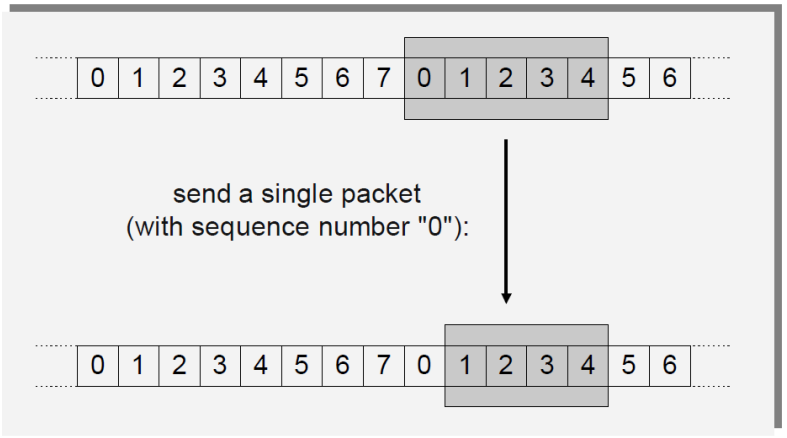
# Sliding window protocol(Cont'd)


- **Receiving Window:**
  - The receiver maintains a *receiving window* corresponding to the sequence numbers of frames that are accepted

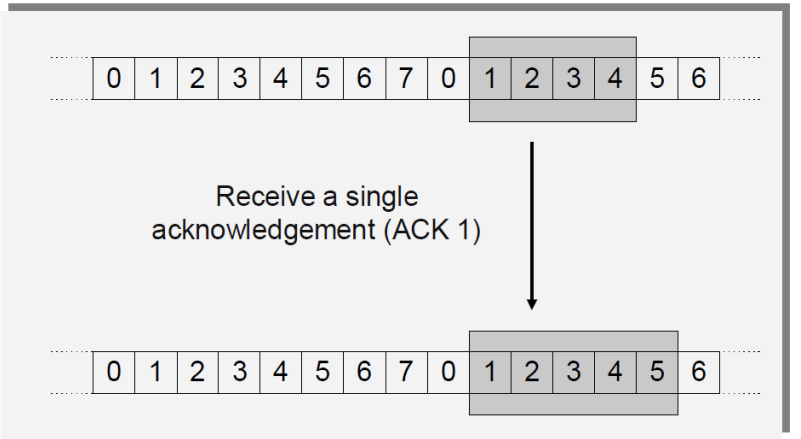


# Sliding window protocol(Cont'd)


- **Operations at the sender:**

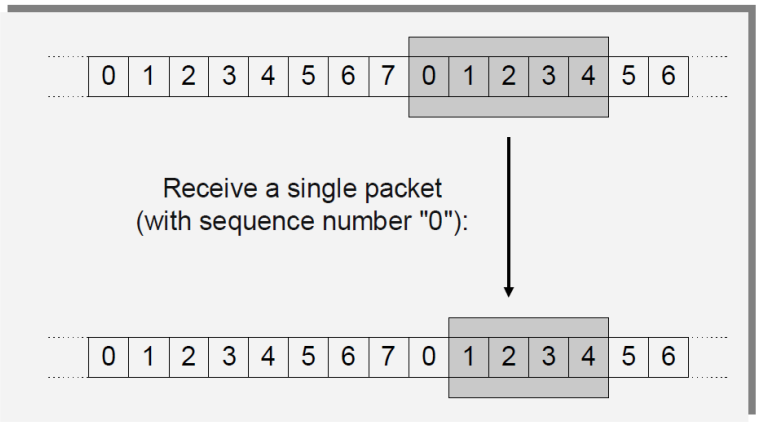



- **Operations at the sender:**

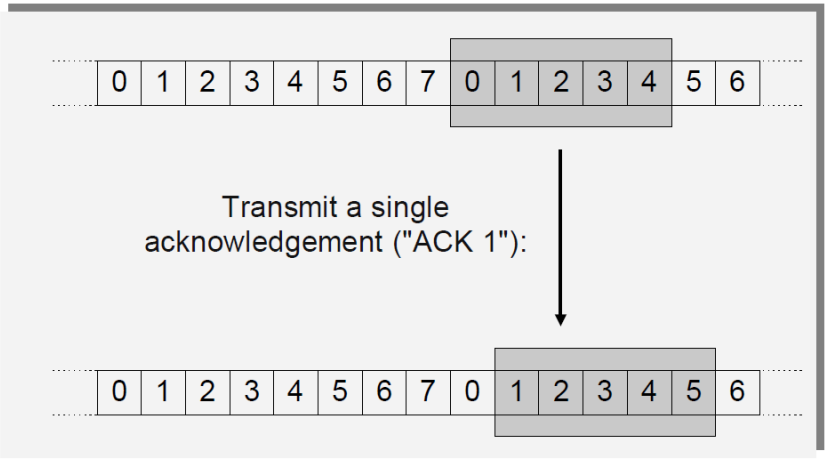


# Sliding window protocol(Cont'd)


- **Operations at the receiver:**

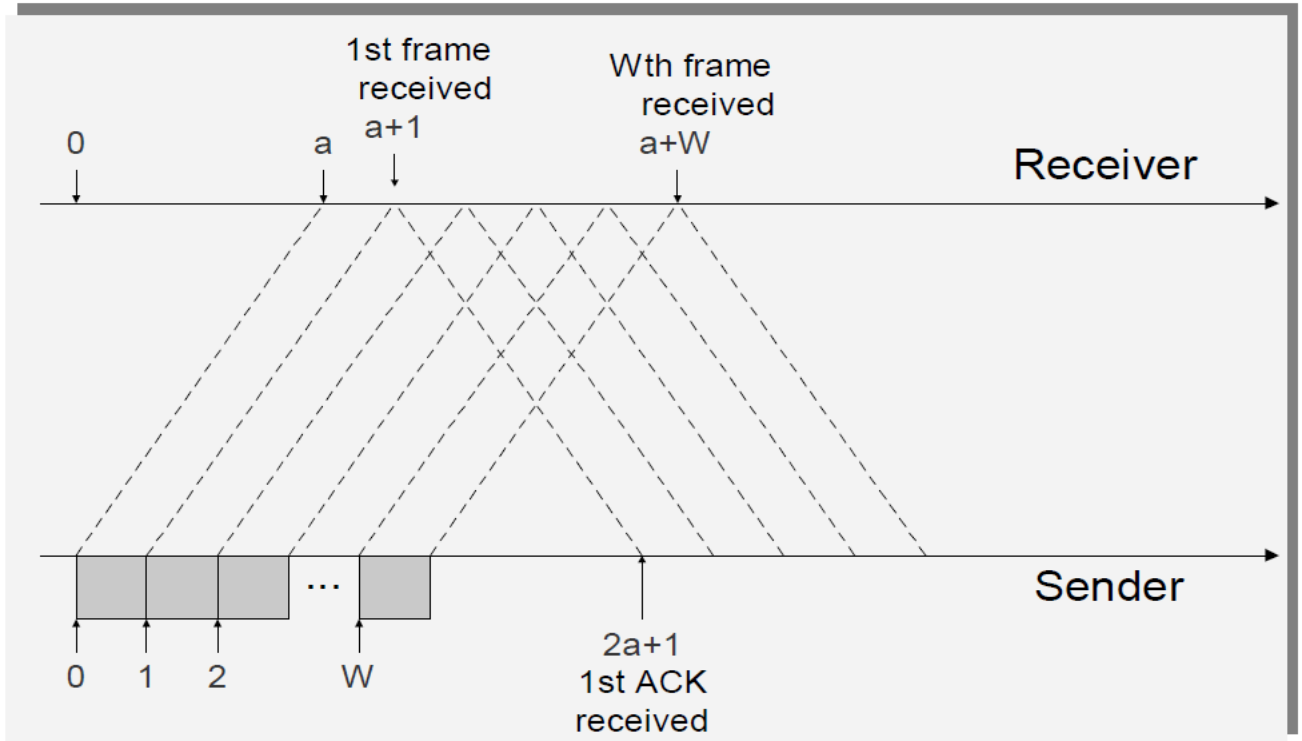



- **Operations at the receiver:**



# Sliding window protocol(Cont'd)

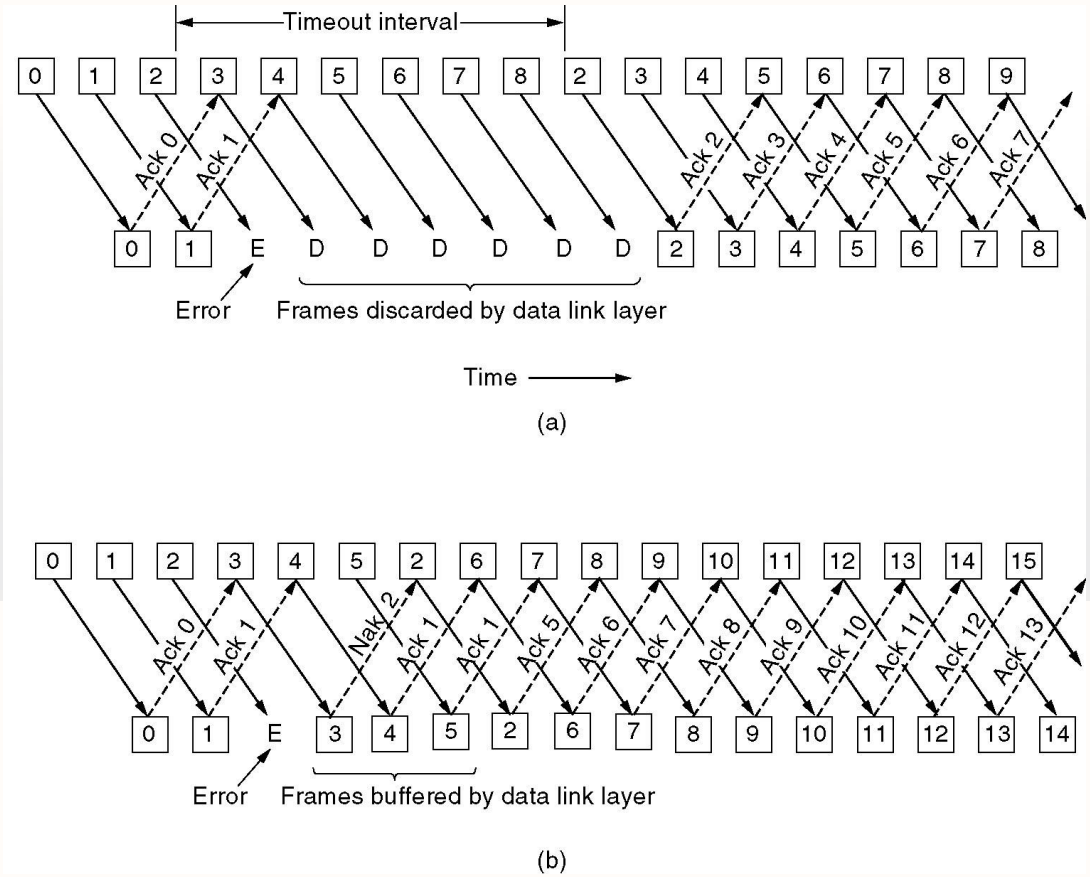

## *Analysis of Sliding Windows*



# Sliding window protocol(Cont'd)


- **Two types of errors:**
- – Lost frames
- – Damaged Frames
- Most Error Control techniques are based on (1) Error Detection Scheme (e.g., Parity checks, CRC), and (2) Retransmission Scheme
- Error control schemes that involve error detection and
- retransmission of lost or corrupted frames are referred to as ***Automatic Repeat Request (ARQ)*** error control

# Sliding window protocol(Cont'd)




- Go-Back-N uses the sliding window flow control protocol. If no errors occur the operations are identical to Sliding Window
- **Operations:**
  - A station may send multiple frames as allowed by the window size
  - Receiver sends a ***NAK i*** if frame ***i*** is in error. *After that, the receiver discards all incoming frames until the frame in error was correctly retransmitted*
  - *If sender receives a **NAK i** it will retransmit frame **i** and all packets **i+1, i+2,...** which have been sent, but not been acknowledged*

# THANK YOU!

Your Logo