Computer Network
(CSE 3034)

Text book: Computer Networks by Andrew S. Tanenbaum

Introduction to the course

Syllabus :

- Introduction(Chapter 1)
- The Physical Layer(Chapter 2)
- **The Data Link Layer(Chapter 3)**
- The Medium Access Control Sublayer(Chapter 4)
- The Network Layer(Chapter 5)
- The Transport layer(Chapter 6)
- The Application layer(Chapter 7)
- Network security(Chapter 8)

The Data Link Layer

The Data Link Layer

- Objective : To achieve reliable, efficient communication between two adjacent machines
- Communication circuits make errors during transmission of bits.
  - Requires error control
- Design issues with data link layer
- Protocols

Data Link Layer Design Issues

Functions of the Data Link Layer :

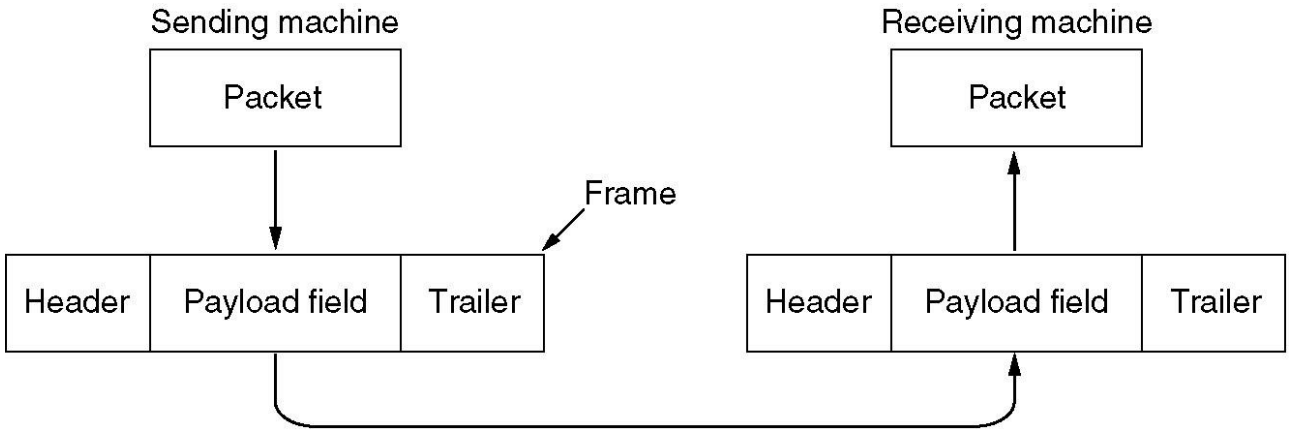
- Provide service interface to the network layer
- Dealing with transmission errors
- Regulating data flow so that slow receivers not swamped by fast senders

Data Link layer :

- Takes the packets from network layer, and encapsulates them into **frames** for transmission using physical layer (reverse process in reception).

Data frame :

1. A frame header
2. A payload field for holding the packet
3. A frame trailer



Relationship between packets and frames

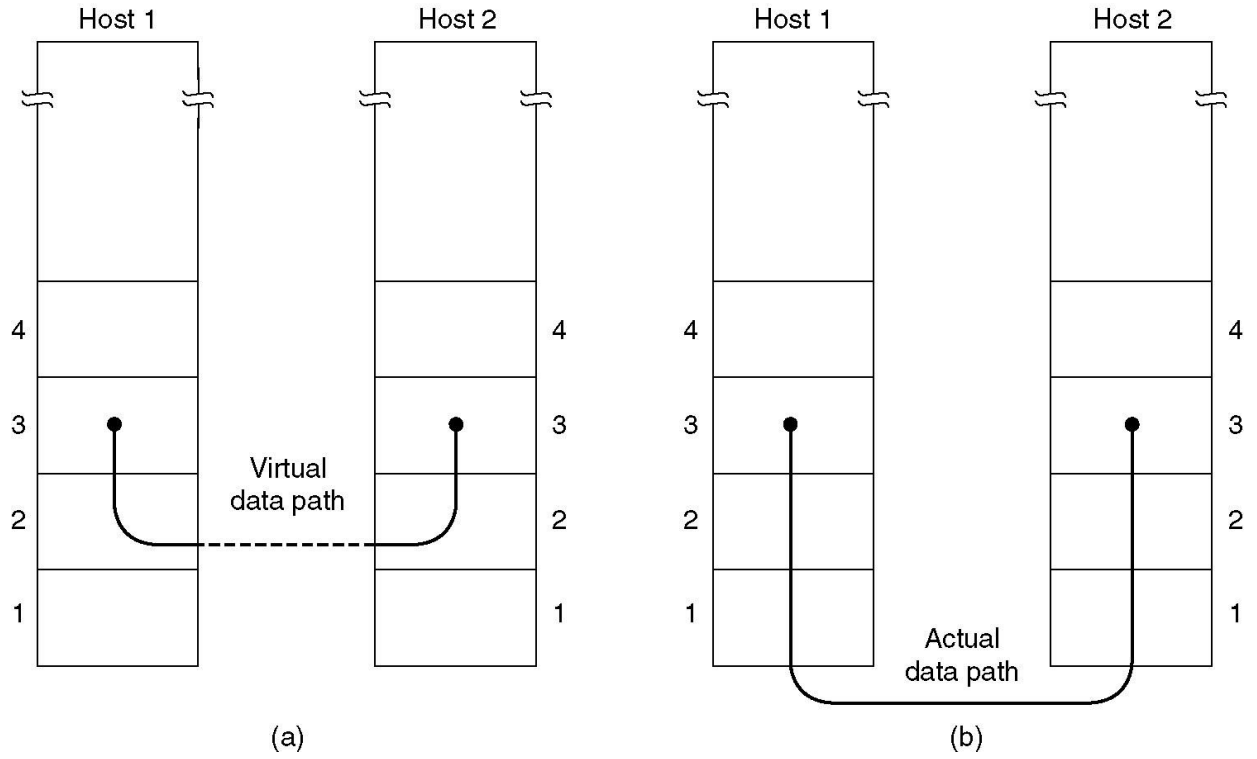
Data Link Layer Design Issues (cont.)

Services Provided to the Network Layer

The job of the data link layer is to transmit the bits obtained from network layer of source machine to the network layer of the destination machine.

Possible services offered by DLL (Data Link Layer) :

- Unacknowledged connectionless service.
- Acknowledged connectionless service.
- Acknowledged connection-oriented service.



(a) Virtual communication. (b) Actual communication.

Data Link Layer Design Issues (cont.)

Services Provided to the Network Layer

Unacknowledged connectionless service :

- Consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.
- No logical connection is established beforehand or released afterward.
- 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.
- Service is appropriate when the error rate is very low so that recovery is left to higher layers.
- Acceptable for real-time traffic, such as voice, in which late data are worse than bad data
- Example: Ethernet, Voice over IP, etc.

Data Link Layer Design Issues (cont.)

Services Provided to the Network Layer

Acknowledged Connectionless Service :

- Each frame send by the Data Link layer is acknowledged and the sender knows if a specific frame has been received or lost.
- Typically the protocol uses a specific time period that if has passed without getting acknowledgment it will re-send the frame.
- This service is useful for commutation when an unreliable channel is being utilized (e.g., 802.11 WiFi).
- Network layer does not know frame size of the packets and other restriction of the data link layer. Hence it becomes necessary for data link layer to have some mechanism to optimize the transmission.

Data Link Layer Design Issues (cont.)

Services Provided to the Network Layer

Acknowledged Connection Oriented Service :

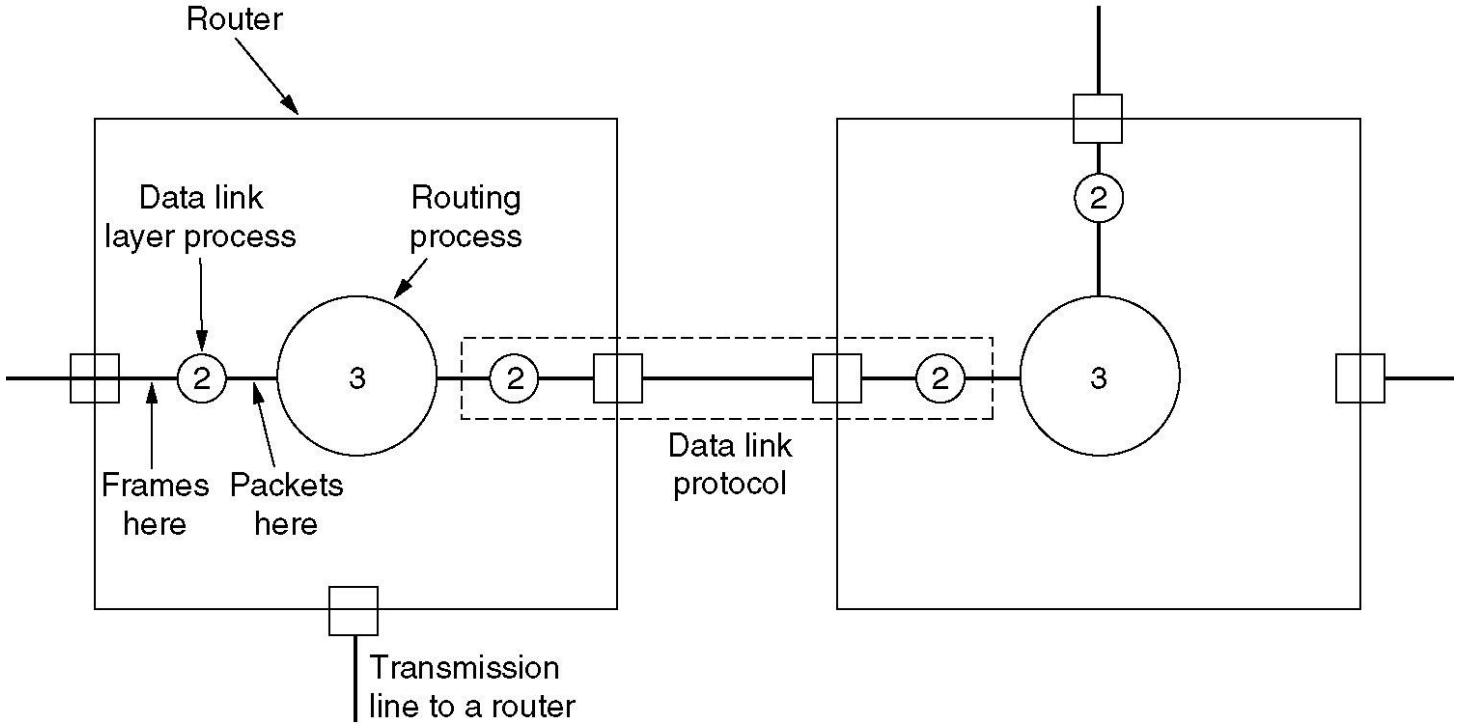
- Source and Destination establish a connection first.
- Each frame sent is numbered
  - Data link layer guarantees that each frame sent is indeed received.
  - It guarantees that each frame is received only once and that all frames are received in the correct order.
- Examples:
  - Satellite channel communication, Long-distance telephone communication, etc.
- Three distinct phases:
  1. Connection is established by having both side initialize variables and counters needed to keep track of which frames have been received and which ones have not.
  2. One or more frames are transmitted.
  3. Finally, the connection is released – freeing up the variables, buffers, and other resources used to maintain the connection.

Data Link Layer Design Issues (cont.)

Services Provided to the Network Layer

Flow over two routers : An example

- When a frame arrives at a router, the hardware checks it for errors, then passes the frame to the data link layer software.
- The data link layer software checks to see if this is the frame expected, and if so, gives the packet contained in the payload field to the routing software.
- The routing software then chooses the appropriate outgoing line and passes the packet back down to the data link layer software, which then transmits it.



Placement of the data link protocol.

Data Link Layer Design Issues (cont.)

Framing

- To provide service to the network layer, the data link layer must use the service provided to it by the physical layer.
- The physical layer deals with transmission of raw bit stream and doesn't give guarantee of error free transmission.
- Errors could be:
  - Number of received bits does not match number of transmitted bits (deletion or insertion)
  - Bit Value
- Data link layer takes the responsibility to detect errors and if possible, to correct errors.
  - It breaks up the bit stream into discrete frames and computes a checksum which it includes with each frame.
  - Braking up the bit stream into frames is more difficult than it seems.

Data Link Layer Design Issues (cont.)

Framing

Framing Methods :

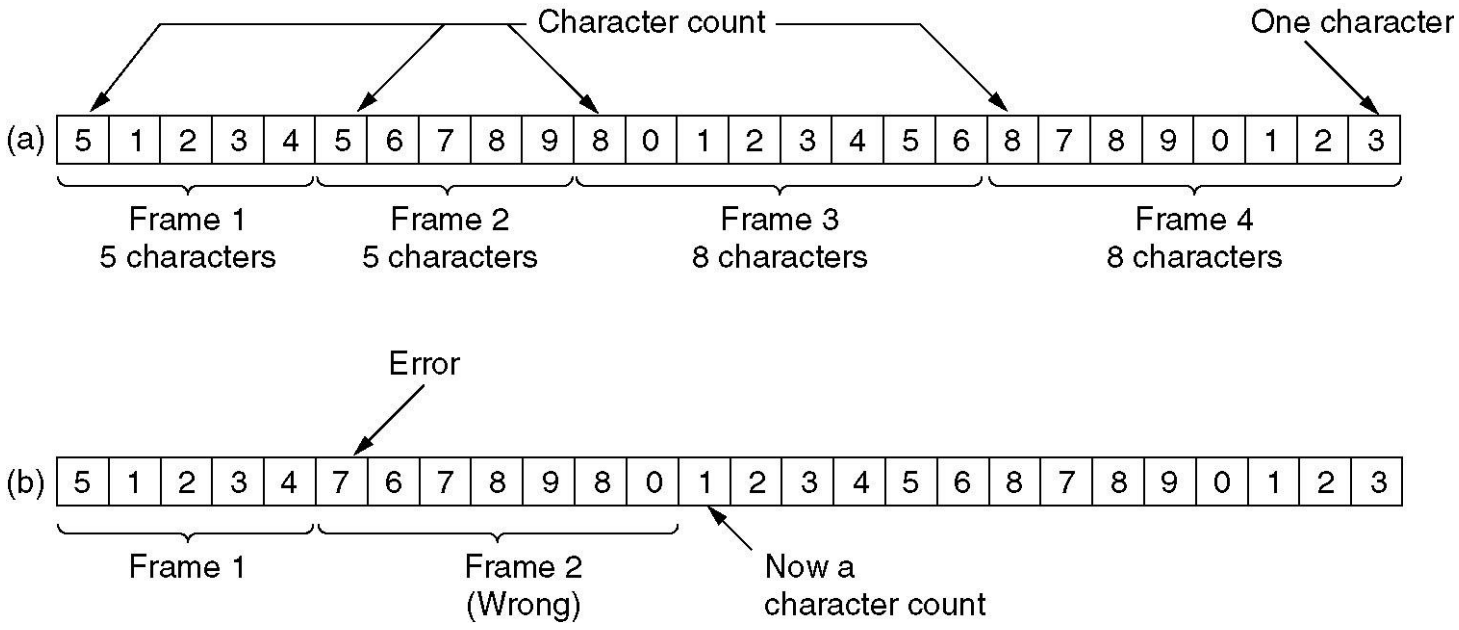
- Character count
- Flag bytes with byte stuffing
- Starting and ending flags with bit stuffing
- Physical layer coding violations

Data Link Layer Design Issues (cont.)

Framing

Character count :

- Uses the first field in the frame's header to indicate the length of the frame, so that the receiver knows how big the current frame is and can determine where does a frame ends.
- Trouble in the algorithm:
  - Receiver loses synchronization when the count become garbled due to transmission error.
  - The receiver will think that the frame contains fewer (or more) characters than it actually does.
- Although checksum will detect the frames are incorrect, the receiver will have difficulty in re-synchronizing to the start of a new frame.



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

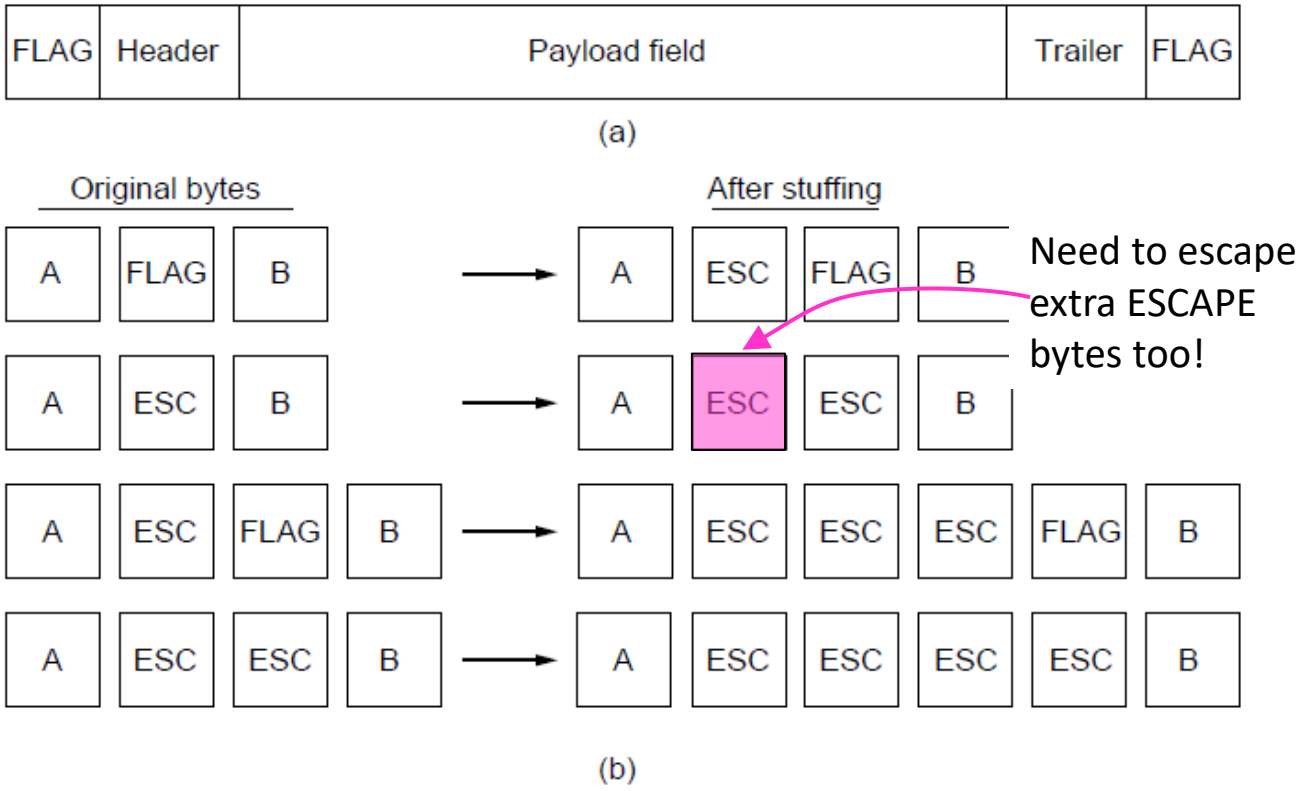
Data Link Layer Design Issues (cont.)

Framing

Flag bytes with byte stuffing :

- Uses frame boundary detection technique though appending of special byte just before and after each frame.
- Most protocols have used same bytes normally called as **flag bytes**.
- Problem occurs when the flag byte value matches a data value.
- Can be solved using an ESC character before flag bytes : known as **character/byte stuffing**.

Limitation : All character codes of 8 bit.



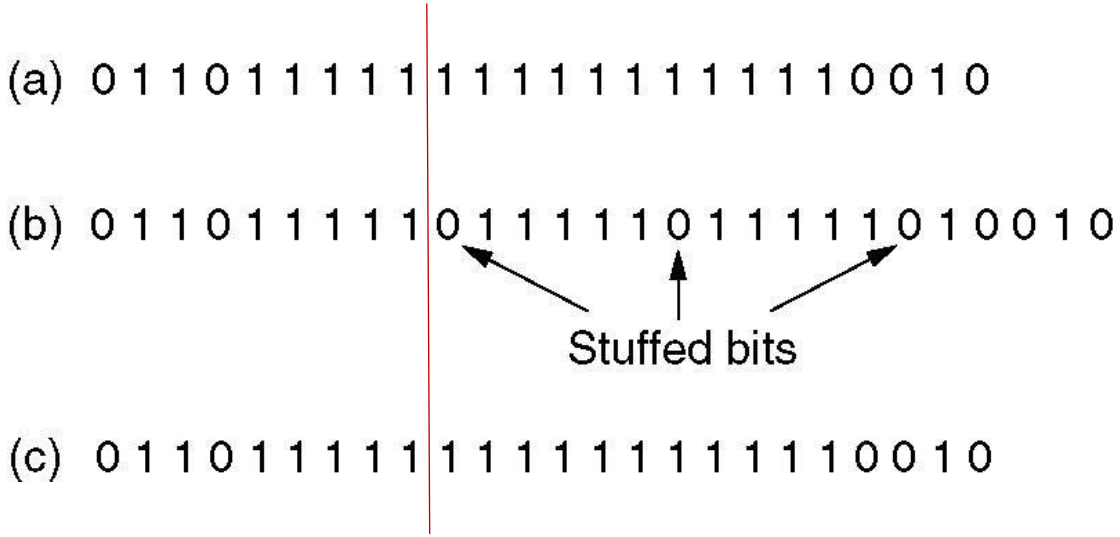
(a) A frame delimited by flag bytes.
(b) Four examples of byte sequences before and after byte stuffing.

Data Link Layer Design Issues (cont.)

Framing

Starting and ending flags with bit stuffing :

- Allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character.
- This methods achieves the same thing as Byte Stuffing method by using Bits (1) instead of Bytes (8 Bits).
- Each frames begins and ends with a special bit pattern (i.e. 011111110 – a flag byte).
- On transmit, after five consecutive 1s in the data, a 0 is added (i.e. **bit stuffing**).
- On receive, a 0 after five 1s is deleted to get original data.
- Flag bytes only occur at frame boundaries, not within the data (i.e. easy to recognize the frame if loses the track while receiving).
- The side effect: the length of a frame depends on the contents of the data it carries.



(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 Link Layer Design Issues (cont.)

Framing

Physical layer coding violations :

- Applicable to networks in which the encoding on the physical medium contains some redundancy.
- Example : Some LANs encode 1 bit of data by using 2 physical bits.
  - Bit ‘1’: high-low
  - Bit ‘0’: low-high

    (high-high and low-low are not used for data)

Note : Many data link protocols use a combination of a character count with one of the other methods for extra safety.

Data Link Layer Design Issues (cont.)

Framing

Q1. The following character encoding is used in a data link protocol:

A: 01000111; B: 11100011; FLAG:01111110; ESC: 11100000

Show the bit sequence transmitted (in binary) for the four-character in a frame: A B ESC FLAG when each of the following framing methods are used:

(a) Character count.

(b) Flag bytes with byte stuffing.

(c) Starting and ending flag bytes, with bit stuffing.

A1 : (a) 00000101 01000111 11100011 11100000 01111110

5 A B ESC FLAG

(b) 01111110 01000111 11100011 11100000 11100000 11100000 01111110 01111110

FLAG A B ESC ESC ESC FLAG FLAG

(c) 01111110 01000111 110100011 111000000 011111010 01111110

Data Link Layer Design Issues (cont.)

Framing

Q2. A bit string, 0111101111101111110, needs to be transmitted at the data link layer. What is the string actually transmitted after bit stuffing?

A2 : The bit string actually transmitted is 011110111110011111010.

Data Link Layer Design Issues (cont.)

Framing

Q3. The following data fragment occurs in the middle of a data stream for which the byte-stuffing algorithm is used: A B ESC C ESC FLAG FLAG D. What is the output after stuffing?

A3 : The output after stuffing is A B ESC ESC C ESC ESC ESC FLAG ESC FLAG D.

Data Link Layer Design Issues (cont.)

Error control

- After solving the marking of the frame with start and end the data link layer must handle eventual errors in transmission.
  - ✓Ensuring that all frames are delivered to the network layer at the destination and in proper order.
- Unacknowledged connectionless service: it is fine for the sender to output frames regardless of its reception.
- Reliable connection-oriented service: it is not fine.
- Usual way to ensure reliable delivery is to include acknowledgement (i.e. special control frames) to the sender.
  - Positive acknowledgement : frame has arrived safely.
  - Negative acknowledgement : something wrong during transmission, so need retransmission
    I. A timer is required to be set so that if the data frame or the acknowledgement is lost then the sender is aware of the problem when the timer runs out.
    II. Sequence numbers required to be associated with frames to ensure the passing of frames to the destination network layer, exactly once in the right order if frames are received more than once.

Data Link Layer Design Issues (cont.)

Flow control

- Important when the sender is running on a fast powerful computer and receiver is running on a slow low-end machine.
- Prevents a fast sender from out-pacing a slow receiver
- Two approaches:
  1. Feedback-based flow control : Can be taken care at the data link layer
     - Receiver gives feedback on the data it can accept and gives permission to send data
  2. Rate-based flow control : Mostly taken care at transport layer
     - Built in mechanism that limits the rate at which sender may transmit data, without the need for feedback from the receiver.

Error Detection and Correction

- Transmission errors: common on local loops (because of analog signalling), wireless links.
- Two basic strategies for dealing with errors.
  - Error correction : Include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been.
    - Applicable to wireless channel , where retransmission can be faulty.
    - Uses error correction codes.

    Ex : Hamming code
  - Error detection : Include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission.
    - Applicable to reliable transmission channel such as optical fiber (requires retransmission)
    - Uses error detection codes.

    Ex : Cyclic Redundancy Code

Error Detection and Correction (cont.)

Handling the error :

A frame consists of **m** data (i.e., message) bits and **r** redundant, or check bits.
Total length be **n** (i.e., **n** = **m**+ **r**) bits (normally known as **codeword**).

Example :

- Transmitted: 10001001
- Received: 10110001

XOR operation gives number of bits that are different.

- XOR: 00111000
- Number of bit positions in which two codewords differ is called ***Hamming Distance***. It shows that two codes are ***d*** distance apart, and it will require ***d*** errors to convert one into the other.

Note :

- All 2^m possible data messages are legal.
- All 2^n possible codewords are not treated as legal as far as the hamming distance and check bits are concerned.

Error Detection and Correction(cont.)

Error correction code

Design of an error correction code requires a significant no. of check bits and has a lower limit defined by the relation

$$(m + r + 1) \leq 2^r \text{ , where 'r' indicates no. of check bits for m message bits.}$$

Example : Hamming code (can be used to correct single error)

- Bits of the codeword : b1 b2 b3 b4 ….
- Check bits: The bits that are powers of 2 (p1, p2, p4, p8, p16, …).
- The rest of bits (m3, m5, m6, m7, m9, …) are filled with ***m*** data bits.
- Check bits are considered as parity bits for subset(i.e. some collection of bits) of the codeword (e.g. p1 is the parity bit for all data bits in positions whose binary representation includes a 1 in the least significant position excluding 1 (3, 5, 7, 9, 11 and so on))
- Data bits are checked by adding together parity bits for its position; e.g., 3 = 1 + 2, 5 = 1 + 4, 6 = 2 + 4, 7 = 1 + 2 + 4, 9 = 1 + 8, 10 = 2 + 8, 11 = 1 + 2 + 8, …

Error Detection and Correction(cont.)

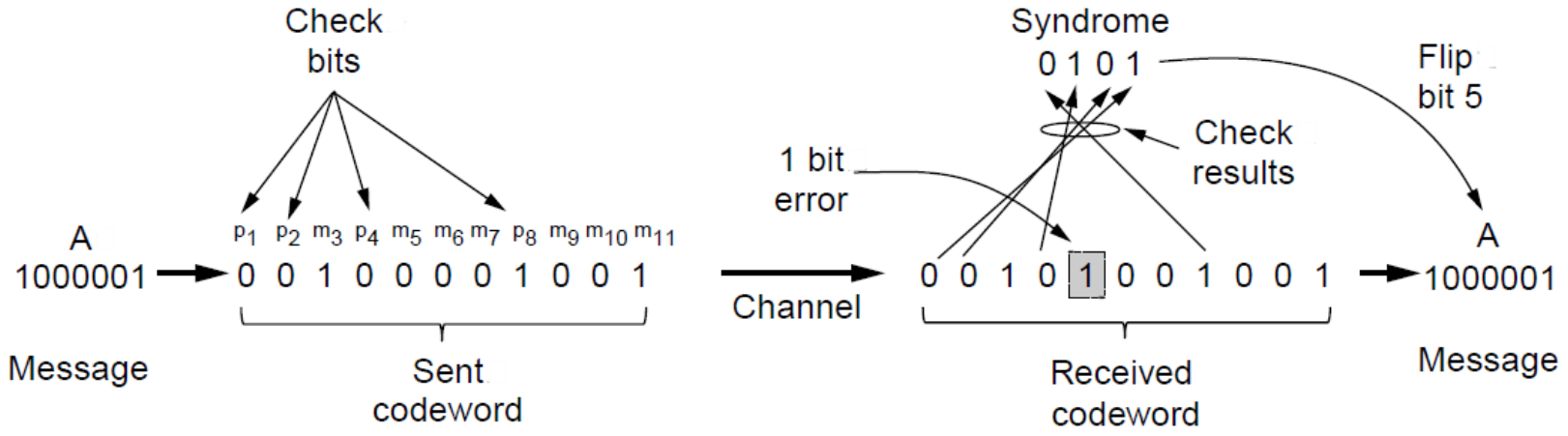
Hamming code

Error correction :

- When a codeword arrives the receiver initializes a counter to '0'.
- After codeword arrives, it examines each check bit ***k*** for correct parity value.
  - If correct the codeword is accepted.
  - If incorrect, it adds ***k*** to the counter and after all check gives the no. of incorrect data bit.

    (e.g. in a (11, 7) codeword, if check bits 1, 2, and 8 are in error, the inverted data bit is 11)
- Recomputing the parity sums (error syndrome) gives the position of the error to flip, or 0 if there is no error.

Example :



An (11, 7) Hamming code correcting a single-bit error

Error Detection and Correction(cont.)

Hamming code

Question :

An 8-bit byte with binary value 10101111 is to be encoded using an even-parity Hamming code. What is the binary value after encoding?

Answer :

Since, $(m + r + 1) \leq 2^r$, $r = 4$ (i.e. p1, p2, p4, p8)

1	2	3	4	5	6	7	8	9	10	11	12	
1	**0**	1	**0**	0	1	0	**0**	1	1	1	1	12-BIT CODEWORD
1	-	1	-	0	-	0	-	1	-	1	-	(EVEN PARITY)
-	**0**	1	-	-	1	0	-	-	1	1	-	(EVEN PARITY)
-	-	-	**0**	0	1	0	-	-	-	-	1	(EVEN PARITY)
-	-	-	-	-	-	-	**0**	1	1	1	1	(EVEN PARITY)

The binary value after encoding is 101001001111.

Error Detection and Correction(cont.)

Error detecting code

- Prefered to be used where the the error rate associated with the transmission line is much lower (e.g. optical fiber, copper wire)
- Usually require retransmission if error is detected.

Say, it is required to transmit 1 megabit of data in the form of data block each of size 1000 bit.

- Using error correction code each block requires 10 check bits for single bit error.
- Requires 10,000 bits to be transmitted for complete data.
- Using error detection code each block requires an extra bit(i.e. parity bit) for single bit error
- Requires an extra block of 1001 bits to be transmitted for complete data.

The total overhead for the error detection + retransmission method is only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

Example : The polynomial code, also known as a CRC (Cyclic Redundancy Check)

Error Detection and Correction(cont.)

Polynomial (or CRC) code

- Bit strings are represented as polynomials with coefficients of 0 and 1(i.e. bit value) only.
- A k-bit frame is regarded as the coefficient list for a polynomial with k terms ranging from x^{k-1} to x^0.

Example:

110001: $1x^5 + 1x^4 + 0x^3 + 0x^2 + 0x^1 + 1x^0$

- Uses modulo 2 arithmetic(i.e. both addition and subtraction are equivalent to exclusive OR)
- The sender and receiver must agree upon a generator polynomial, G(x), in advance.
- Both high- and low-order bits in the generator polynomial must be 1.
- CRC is computed for a frame of length m- bits (i.e. M(x)) using G(x) known as check summed frame, such that M(x) is longer than the G(x).
- The resulting check summed frame must be divisible by G(x).
- When the receiver gets a check summed frame it divides it by G(x).
  - If the result is not equal to zero it means that there has been transmission error.

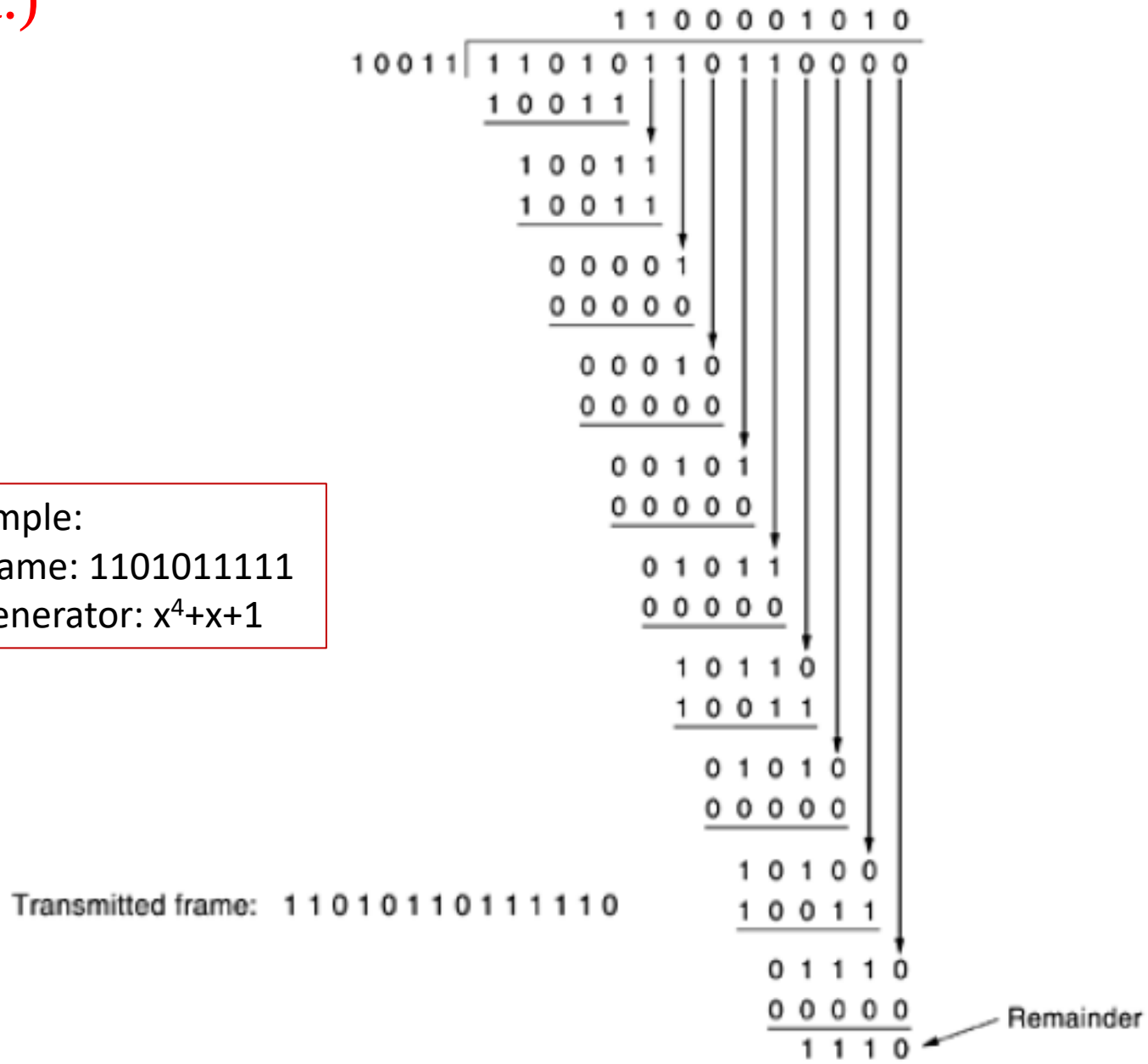
Error Detection and Correction(cont.)

Polynomial (or CRC) code

Algorithm for computing the checksum:

1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m+r bits and corresponds to the polynomial $x^r M(x)$
2. Divide the bit string corresponding to G(x) into the bit string corresponding to $x^r M(x)$ using modulo 2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using module 2 subtraction. The result is the checksummed frame, T(x), to be transmitted.

Example:
Frame: 1101011111
Generator: x^4+x+1



Calculation of the polynomial code checksum

Elementary data link protocols

Assumptions:

1). **DLL** and **Network** layer are independent processes that communicate by passing messages back and forth trough the physical layer.
2). Machine **A** wants to send a long stream of data to machine **B**, using a reliable, **connection-oriented service**.
3). Machines do not crash.

Channel: Noiseless

No frames are lost, duplicated, or corrupted.

Protocols :

- An Unrestricted Simplex Protocol
- A Simplex Stop-and-Wait Protocol

Elementary data link protocols (cont.)

Library procedures:

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

Note : Defined in a header file *protocol.h*

Elementary data link protocols (cont.)

Protocol definitions:

- Five data structures are defined : *boolean, seq_nr, packet, frame_kind,* and *frame*.
- Defined in *protocol.h*

```
#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;
```

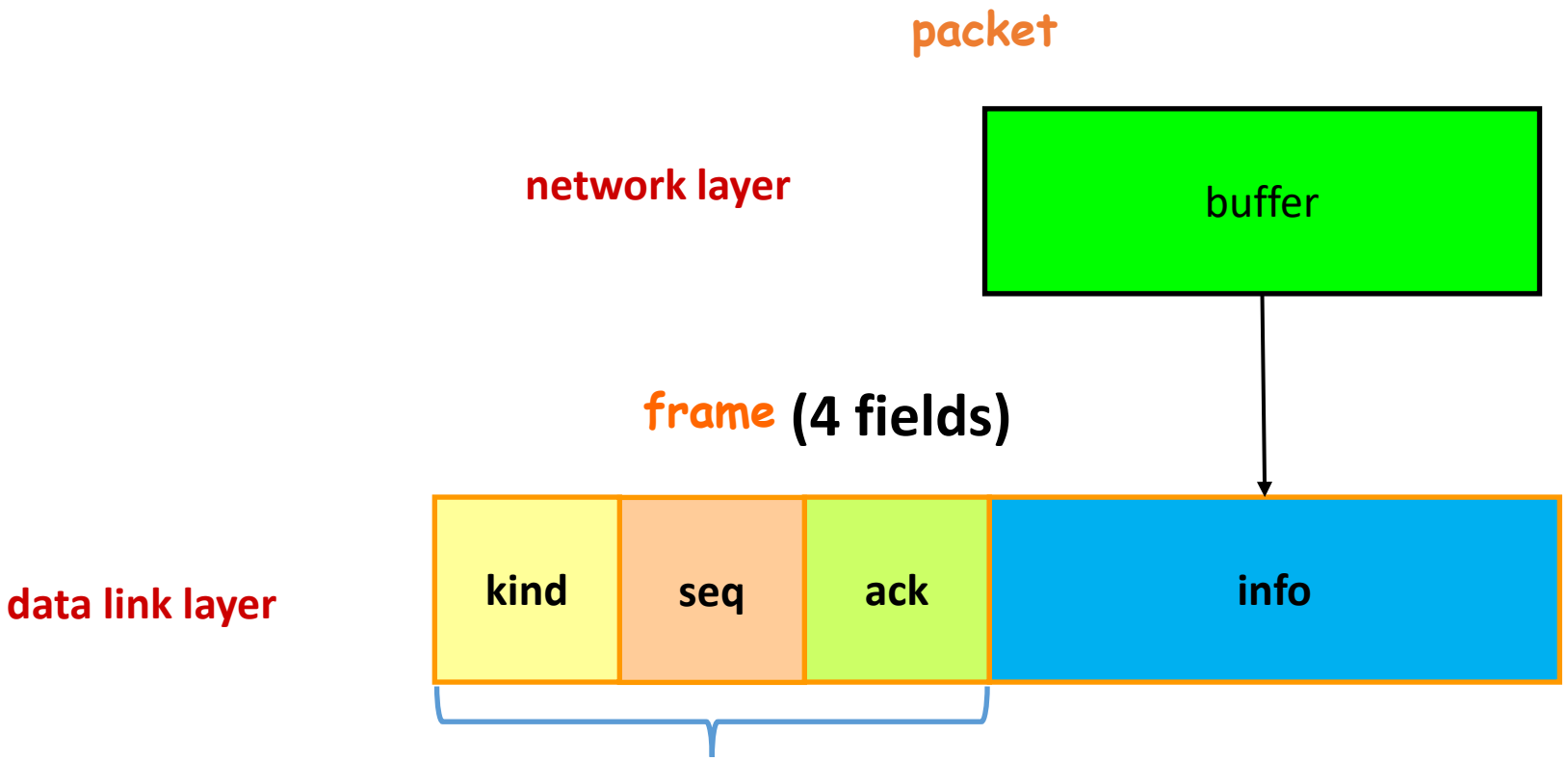
Continued →

Elementary data link protocols (cont.)

Protocol Definitions (ctd.)

```
/* 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
```

Elementary data link protocols (cont.)

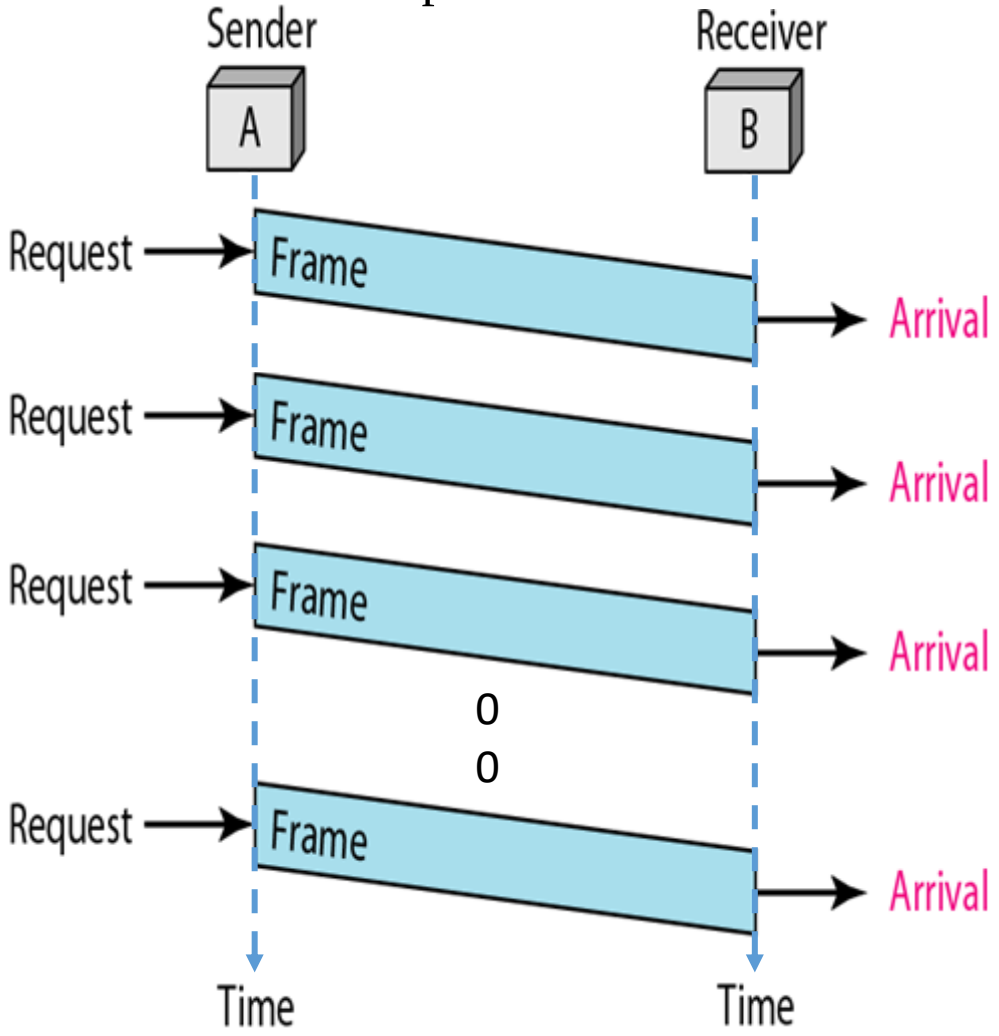


Elementary data link protocols (cont.)

An Unrestricted Simplex Protocol

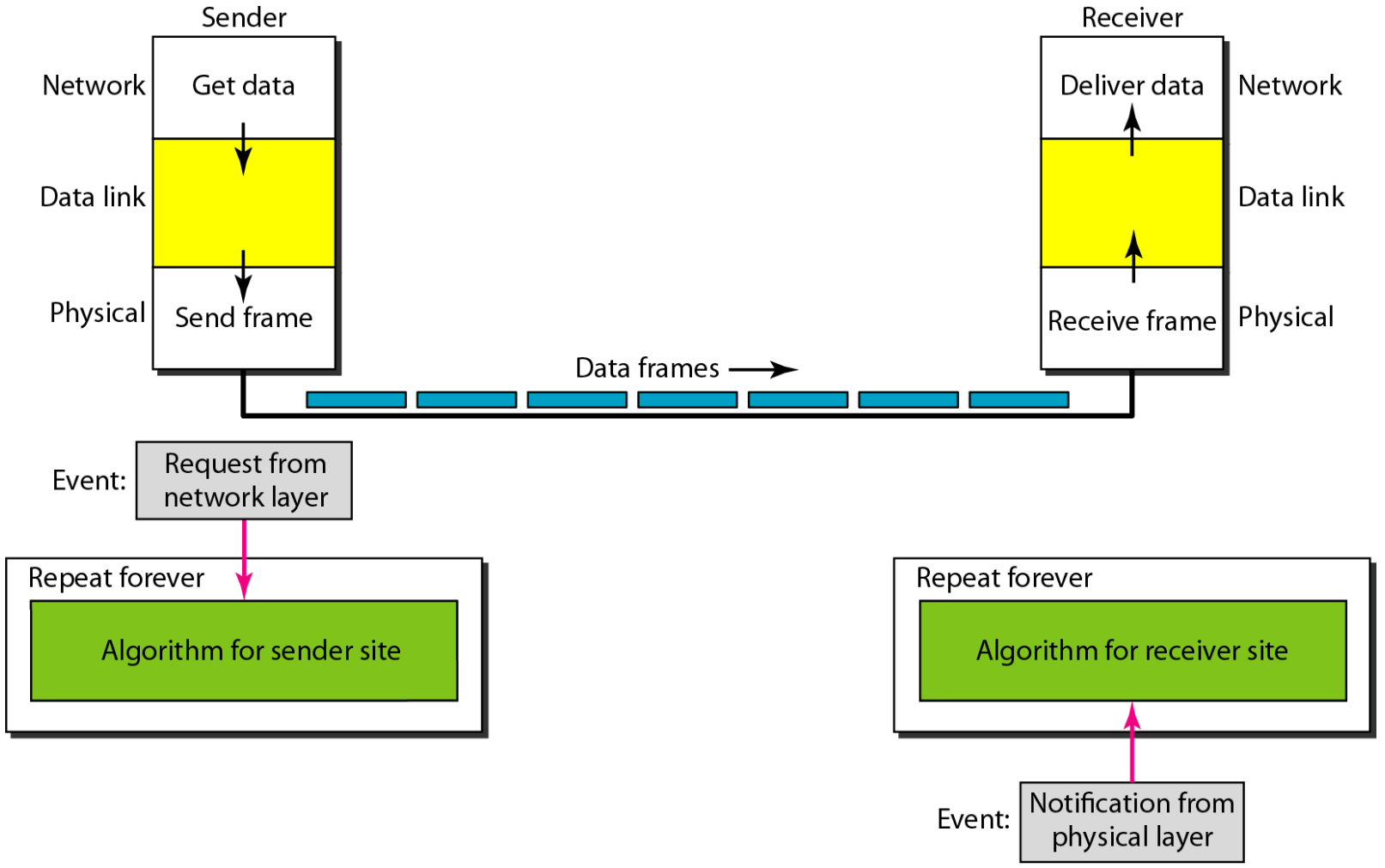
- Simplest Protocol (called as **utopia**)
- Unidirectional in which *data frames are traveling in only one direction*-from the sender to receiver.
- Both transmitting and receiving network layers are always ready.
- The receiver can immediately handle any frame it receives with a processing time that is small enough to be negligible.
- The data link layer of the receiver immediately removes the header from the frame and hands the data packet to network layer.
- In other words, the receiver can never be fill out with incoming frames.
- Unrealistic and resembles to connectionless
- Relies on higher layer for error handling

Communication using (Flow Diagram)
Simplest Protocol



Elementary data link protocols (cont.)

An Unrestricted Simplex Protocol



Design of the utopia protocol with no flow or error control

Elementary data link protocols (cont.)

An Unrestricted Simplex Protocol

Runs at sender machine

Runs at receiver machine

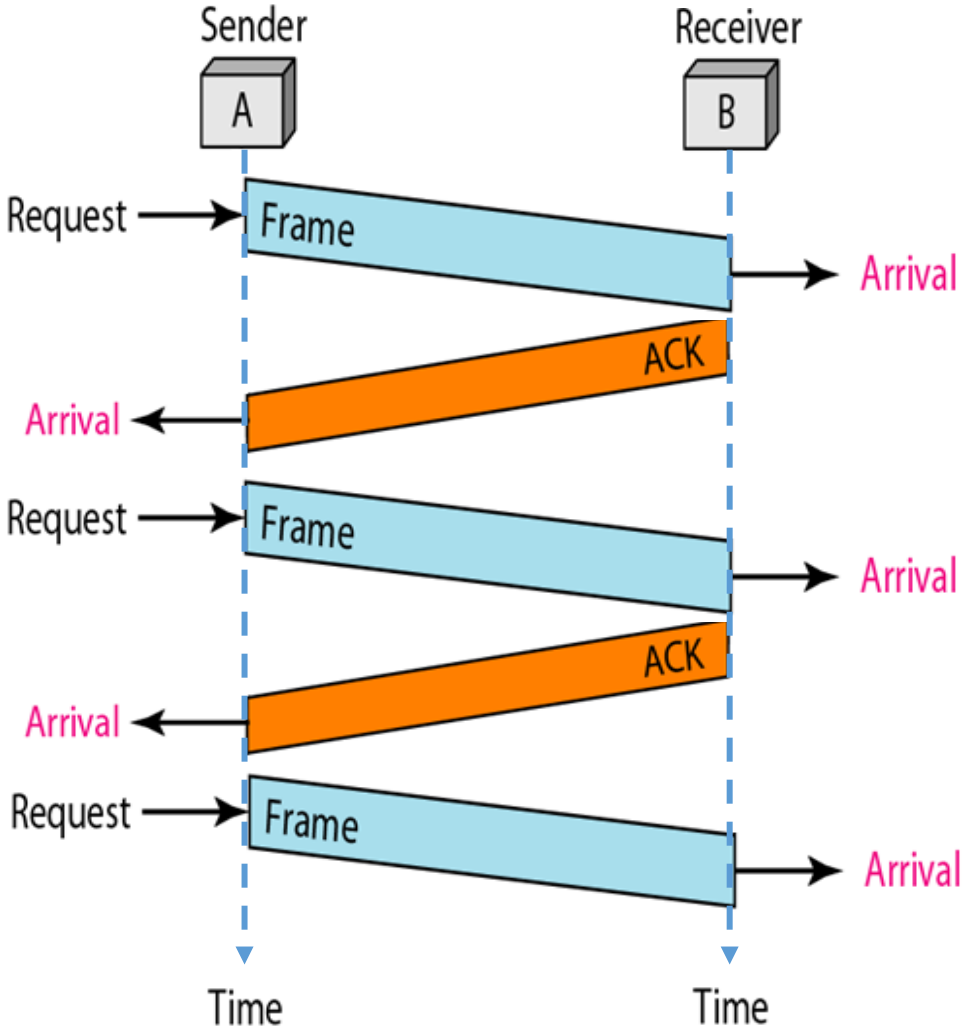
```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free,
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping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

Elementary data link protocols (cont.)

A Simplex Stop-and-wait Protocol

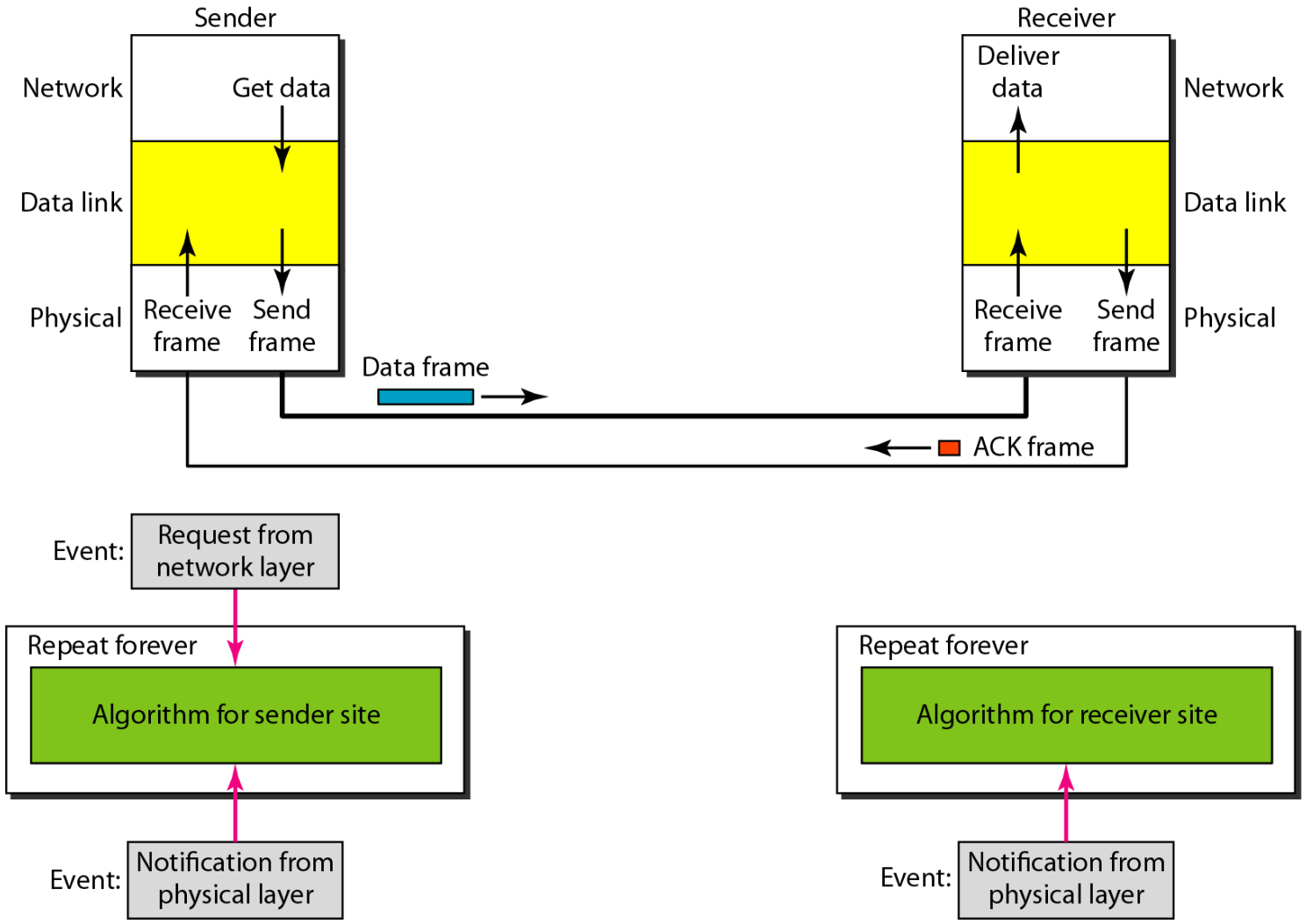
- If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use.
- Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources.
- Required to tell the sender to slow down.
- There must be feedback from the receiver to the sender.
- The sender sends one frame, stops until it receives agreement the receiver (okay to go ahead), and then sends the next frame.
- Still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction.
- Includes flow control

Communication (Flow Diagram) using Stop-and-Wait Protocol



Elementary data link protocols (cont.)

A Simplex Stop-and-wait Protocol



Design of Stop-and-Wait Protocol

Elementary data link protocols (cont.)

A Simplex Stop-and-wait Protocol

Runs at sender machine

Runs at receiver machine

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

Sliding Window Protocols

- In most practical situations, there is a need for transmitting data in both directions.
- One way of achieving full-duplex data transmission is to have **two separate communication channels** and **use each one for simplex data traffic** (in different directions).
- If this is done, we have two separate physical circuits, each with a ''forward'' channel (for data) and a ''reverse'' channel (for acknowledgements).
- **In both cases the bandwidth of the reverse channel is almost entirely wasted**. In effect, the user is paying for two circuits but using only the capacity of one.

Sliding Window Protocols

Improvement…

- A better idea is to **use the same circuit for data in both directions**.
- In this model the **data frames from A to B are intermixed with the acknowledgement frames from A to B**.
- By looking at the **kind field** in the **header of an incoming frame**, the receiver can tell **whether the frame is data or acknowledgement**.

Sliding Window Protocols
Piggybacking

- When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet.
- The **acknowledgement is attached to the outgoing data frame** (using the **ack field** in the frame header).
- In effect, the *acknowledgement gets a free ride* *on the next outgoing data frame.*
- The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**.

Sliding Window Protocols
Piggybacking

- The principal **advantage** of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth.
- The **ack field** in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum.
- However, piggybacking introduces a complication not present with separate acknowledgements. **How long should the data link layer wait for a packet onto which to piggyback the acknowledgement**?
- **If a new packet arrives quickly, the acknowledgement is piggybacked onto it; otherwise, if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame.**

Sliding Window Protocols

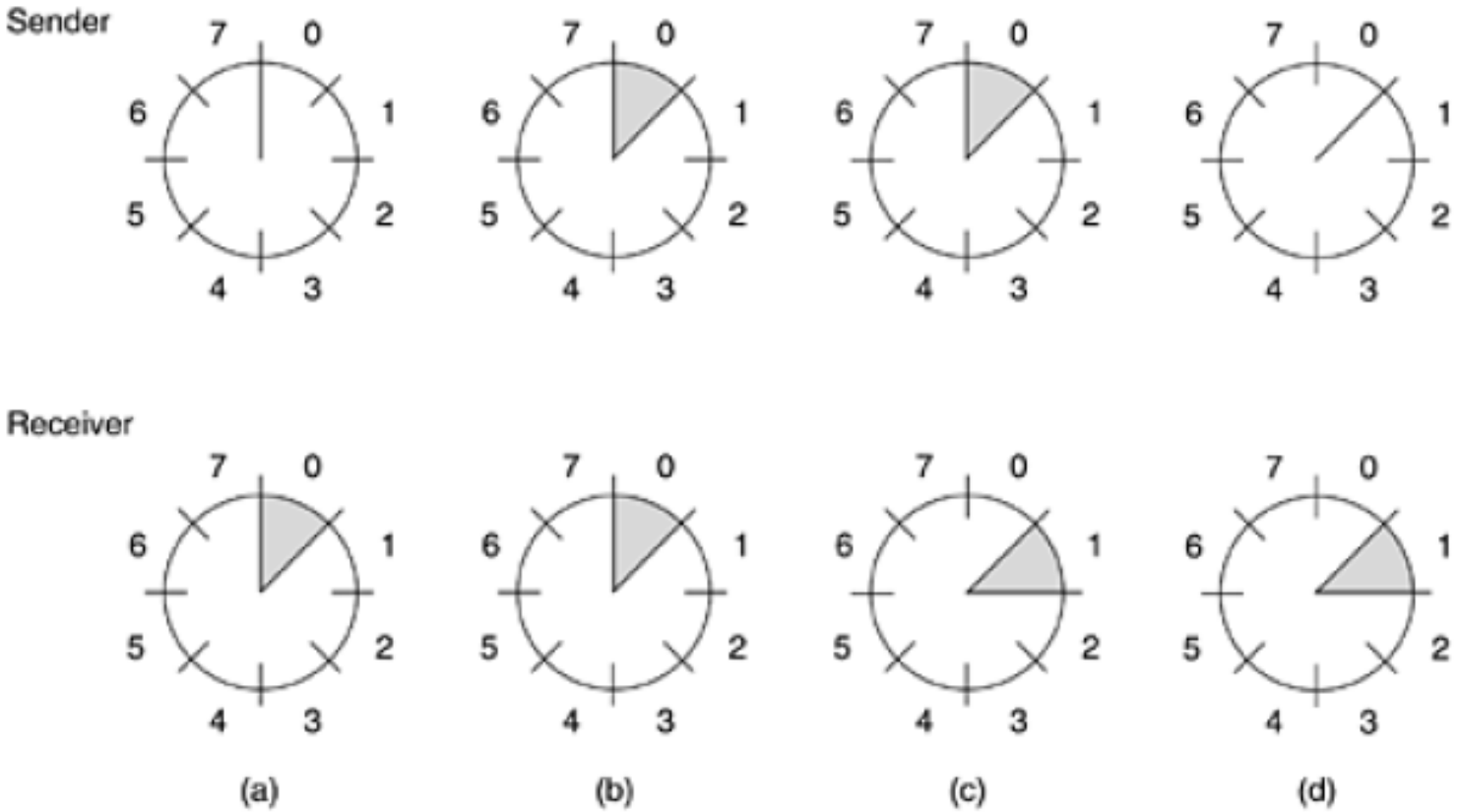
- All sliding window protocols, each **outbound frame** contains a **sequence number**, ranging from **0** up to some **maximum**.
- The **maximum** is usually $\mathbf{2^n - 1}$ so the sequence number fits exactly in an n-bit field.
- The **stop-and-wait sliding window protocol** uses **n = 1**, restricting the sequence numbers to 0 and 1, but more sophisticated versions can use arbitrary n.

Sliding Window Protocols

- The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the sending window.
- Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept.
- The sender's window and the receiver's window need not have the same lower and upper limits or even have the same size.
- In some protocols they are fixed in size, but in others they can grow or shrink over the course of time as frames are sent and received.

Sliding Window Protocols

Figure 3-13. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.



A One-Bit Sliding Window Protocol
Sliding Window Protocols

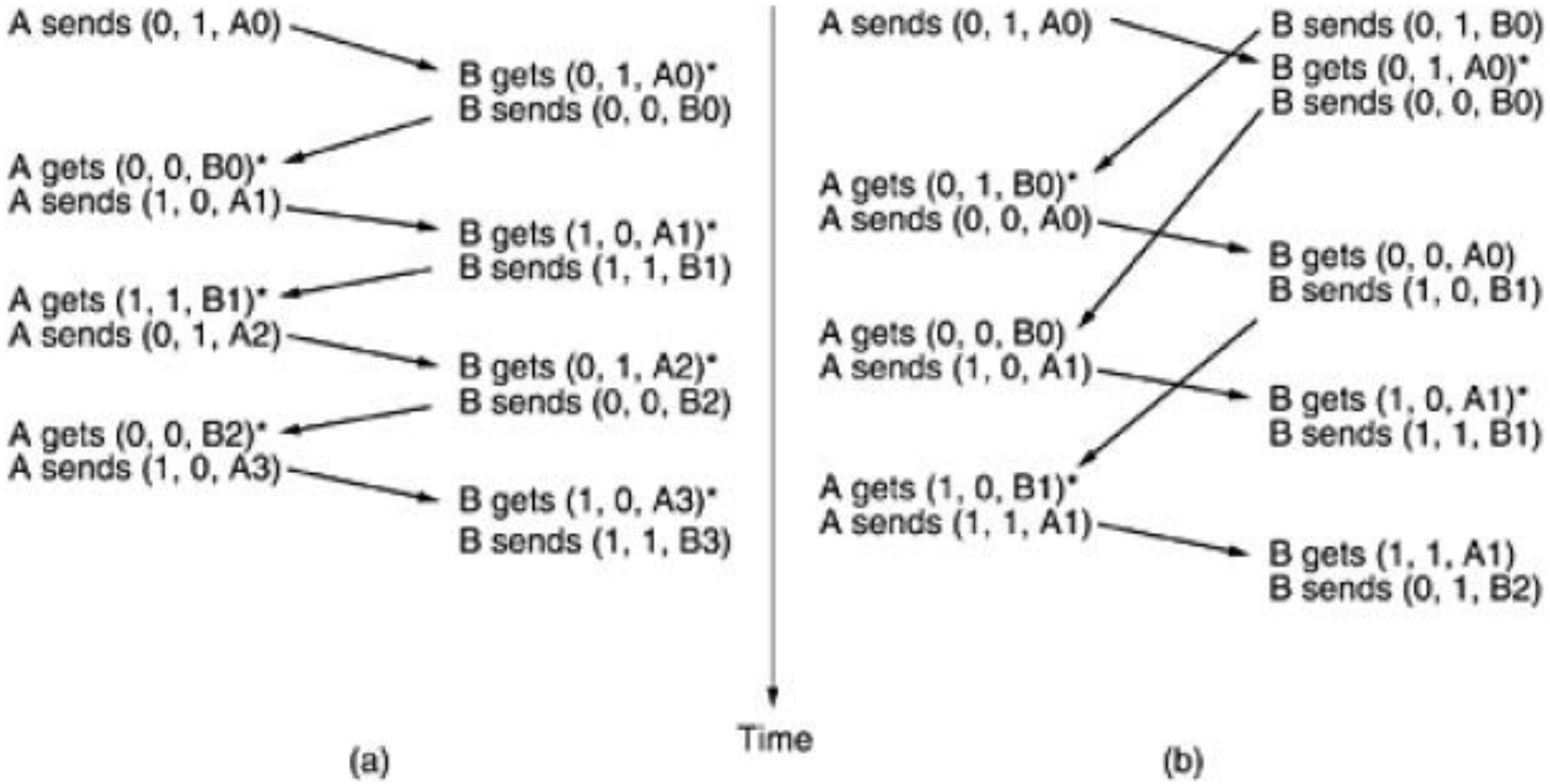
```
/* Protocol 4 (sliding window) is bidirectional. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 0 or 1 only */
  seq_nr frame_expected;                            /* 0 or 1 only */
  frame r, s;                                       /* scratch variables */
  packet buffer;                                    /* current packet being sent */
  event_type event;

  next_frame_to_send = 0;                           /* next frame on the outbound stream */
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network layer */
  s.info = buffer;                                  /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                       /* insert sequence number into frame */
  s.ack = 1 – frame_expected;                       /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
```

A One-Bit Sliding Window Protocol
Sliding Window Protocols

```
while (true) {
    wait_for_event(&event);                 /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) {           /* a frame has arrived undamaged. */
        from_physical_layer(&r);            /* go get it */
        if (r.seq == frame_expected) {      /* handle inbound frame stream. */
            to_network_layer(&r.info);      /* pass packet to network layer */
            inc(frame_expected);            /* invert seq number expected next */
        }
        if (r.ack == next_frame_to_send) {  /* handle outbound frame stream. */
            stop_timer(r.ack);              /* turn the timer off */
            from_network_layer(&buffer);    /* fetch new pkt from network layer */
            inc(next_frame_to_send);        /* invert sender's sequence number */
        }
    }
    s.info = buffer;                        /* construct outbound frame */
    s.seq = next_frame_to_send;             /* insert sequence number into it */
    s.ack = 1 - frame_expected;             /* seq number of last received frame */
    to_physical_layer(&s);                  /* transmit a frame */
    start_timer(s.seq);                     /* start the timer running */
  }
}
```

A One-Bit Sliding Window Protocol
Sliding Window Protocols

Figure 3-15. Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.



A Protocol Using Go Back N
Sliding Window Protocols

- Until now we have made the tacit assumption that the transmission time required for a frame to arrive at the receiver plus the transmission time for the acknowledgement to come back is **negligible**.
- Sometimes this assumption is clearly false.
- In these situations, the **long round-trip time can have important implications** for the **efficiency of the bandwidth utilization**.

A Protocol Using Go Back N
Sliding Window Protocols

- As an example, consider a 50-kbps satellite channel with a 500-msec round-trip propagation delay. Let us imagine trying to use protocol 4 to send 1000-bit frames via the satellite. At t= 0 the sender starts sending the first frame. At t = 20 msec the frame has been completely sent.
- Not until t = 270 msec has the frame fully arrived at the receiver, and not until t = 520 msec has the acknowledgement arrived back at the sender, under the best of circumstances (no waiting in the receiver and a short acknowledgement frame).
- This means that the sender was blocked during 500/520 or 96 percent of the time. In other words, **only 4 percent of the available bandwidth** was used. Clearly, **the combination of a long transit time, high bandwidth, and short frame length is disastrous in terms of efficiency**.

A Protocol Using Go Back N
Sliding Window Protocols

- The **problem** described above can be viewed because of the rule requiring a sender to wait for an acknowledgement before sending another frame.
- If we relax that restriction, much better efficiency can be achieved.
- Basically, the **solution** lies in allowing the sender to transmit up to w frames before blocking, instead of just 1.
- With an appropriate choice of w the sender will be able to continuously transmit frames for a time equal to the round-trip transit time without filling up the window.

A Protocol Using Go Back N
Sliding Window Protocols

- In the example above, $\boldsymbol{w}$ should be at least **26**.
- The sender begins sending frame 0 as before. By the time it has **finished sending 26 frames, at t = 520**, **the acknowledgement for frame 0 will have just arrived**.
- Thereafter, acknowledgements arrive every 20 msec, so the sender always gets permission to continue just when it needs it. At all times, 25 or 26 unacknowledged frames are outstanding. Put in other terms, the sender's maximum window size is 26.

A Protocol Using Go Back N
Sliding Window Protocols

- This technique is known as **pipelining**.

Channel capacity is $\boldsymbol{b\ bits/sec}$,

Frame size $\boldsymbol{l\ bits}$,

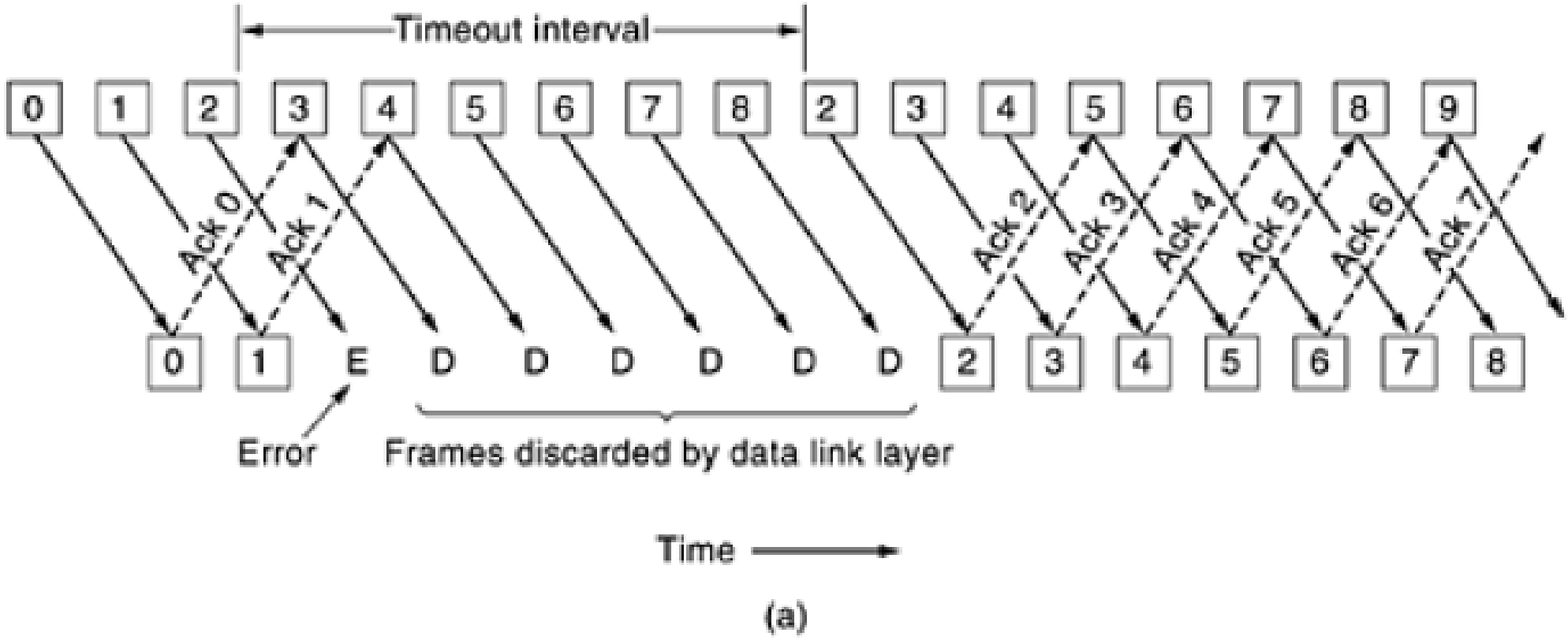
Roundtrip propagation time $\boldsymbol{R\ sec}$,

Then, the time required to transmit a single frame is $\boldsymbol{l/b\ sec}$.

$$line\ utilization = l/(l + bR)$$

A Protocol Using Go Back N
Sliding Window Protocols

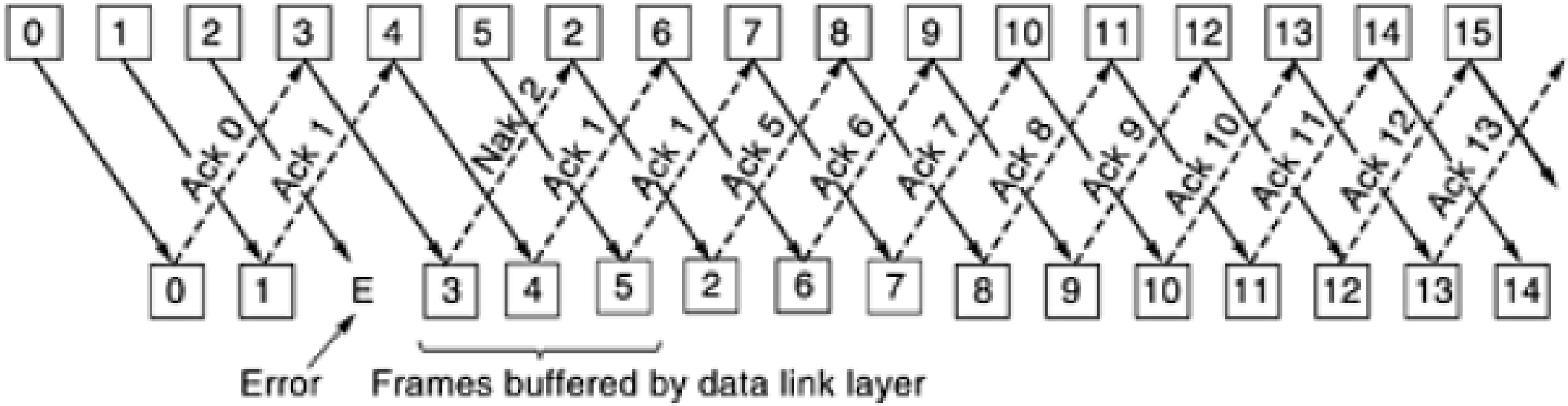
- Pipelining and error recovery.
- Effect of an error when (a) receiver's window size is 1



(a)

A Protocol Using Go Back N
Sliding Window Protocols

- Pipelining and error recovery.
- Effect of an error when (b) receiver's window size is large



(b)

A Protocol Using Selective Repeat
Sliding Window Protocols

- **Go back N** works well if errors are rare, but if the *line is poor, it wastes a lot of bandwidth on retransmitted frames*.
- An alternative strategy for handling errors is to allow the receiver to accept and buffer the frames following a damaged or lost one. Such a protocol does not discard frames merely because an earlier frame was damaged or lost.
- In this protocol, both sender and receiver maintain a window of acceptable sequence numbers.
- The sender’s window size starts out at 0 and grows to some predefined maximum, MAX_SEQ. The receiver's window, in contrast, is always fixed in size and equal to MAX_SEQ.

A Protocol Using Selective Repeat
Sliding Window Protocols

- The receiver has a **buffer reserved** for each sequence number within its fixed window. Associated with each buffer is a bit (arrived) telling whether the **buffer is full or empty**.
- Whenever a frame arrives, its sequence number is checked by the function between to see if it falls within the window. If so and if it has not already been received, it is accepted and stored. This action is taken without regard to whether it contains the next packet expected by the network layer.
- It must be **kept within the data link layer** and not passed to the network layer until all the **lower-numbered frames** have already been delivered to the network layer in the **correct order**.

Error Detection and Correction(cont.)

The error-detecting and error-correcting properties of a code depend on this Hamming distance.

- To reliably detect ***d*** error, one would need a distance ***d*+1** code.

  Ex : Forming a code with adding a parity bit while transmitting can be used to detect single error even though it has a hamming distance of 2.

  (when 1011010 transmitted with even parity the code will be 10110100. A single bit error produces a code word of odd parity)

- To correct ***d*** error, one would need a distance 2***d*+1** code.

  Ex : 4 valid codes: 0000000000, 0000011111, 1111100000, 1111111111

  Minimal Distance of this code is 5 => can correct double errors.

  (If the codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111)

Error Detection and Correction(cont.)

Hamming code

Consider a message having four data bits (D) which is to be transmitted as a 7-bit codeword by adding three check bits. This would be called a (7,4) code. The three bits to be added are three EVEN Parity bits (P), where the parity of each is computed on different subsets of the message bits as shown below.

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

- For example, the message 1101 would be sent as 1010101, since:

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

Problem

- Let us assume that m = 3 and n = 4. Find the list of valid data words and codewords assuming the check bit is used to indicate even parity in the code word.

Valid dataword	Valid codeword
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

Problem

What is the Hamming distance for each of the following codewords:

a. (10000, 00000)

b. (10101, 10000)

c. (11111,11111)

d. (000, 000)

Ans:

a. 1

b. 2

c. 0

d. 0

Problem

Find the minimum Hamming distance to be implemented in codeword for the following cases:

a. Detection of two errors.

b. Correction of two errors.

c. Detection of 3 errors or correction of 2 errors.

d. Detection of 6 errors or correction of 2 errors.

Problem

Find the minimum Hamming distance to be implemented in codeword for the following cases:

a. Detection of two errors.

b. Correction of two errors.

c. Detection of 3 errors or correction of 2 errors.

d. Detection of 6 errors or correction of 2 errors.

a. For error detection → Hamming distance = d + 1 = 2 + 1 = 3

b. For error correction → Hamming distance = 2d + 1 = 2 × 2 + 1 = 5

c. For error detection → Hamming distance = d + 1 = 3 + 1 = 4

For error correction → Hamming distance = 2d + 1 = 2 × 2 + 1 = 5

Therefore, minimum Hamming distance should be 5.

d. For error detection → Hamming distance = d + 1 = 6 + 1 = 7

For error correction → Hamming distance = 2d + 1 = 2 × 2 + 1 = 5

Therefore, minimum Hamming distance should be 7.

Problem

Given in the table a set of valid dataword and codeword.

What is the dataword transmitted for the following codewords received assuming there is 1 bit error?

a. 01010

b. 11010

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Problem

Given in the table a set of valid dataword and codeword.

What is the dataword transmitted for the following codewords received assuming there is 1 bit error?

a. 01010

b. 11010

Answer:

a. 01

b. 11

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Problem

- Sixteen-bit messages are transmitted using a Hamming code. How many check bits are needed to ensure that the receiver can detect and correct single bit errors? Show the bit pattern transmitted for the message 1101001100110101. Assume that even parity is used in the Hamming code.

Problem

- Sixteen-bit messages are transmitted using a Hamming code. How many check bits are needed to ensure that the receiver can detect and correct single bit errors? Show the bit pattern transmitted for the message 1101001100110101. Assume that even parity is used in the Hamming code.

Answer:

- 5 check bits are needed at positions 1, 2, 4, 8, and 16.
- The bit pattern transmitted for the message (first)1101001100110101 is 011010110011001110101

Problem

- An 8-bit message using even-parity Hamming code is received as 101001001111. Find the 8-bit message after getting decoded assuming no error during transmission?

Problem

- An 8-bit message using even-parity Hamming code is received as 101001001111. Find the 8-bit message after getting decoded assuming no error during transmission?

- Answer:

The 8-bit message after decoding is 10101111.

Problem

- A 12-bit Hamming code whose hexadecimal value is 0xE4F arrives at a receiver. What was the original value in hexadecimal? Assume that not more than 1 bit is in error.

Problem

- A 12-bit Hamming code whose hexadecimal value is 0xE4F arrives at a receiver. What was the original value in hexadecimal? Assume that not more than 1 bit is in error.

Answer:

- If we number the bits from left to right starting at bit 1, in this example bit 2 (a parity bit) is incorrect. The 12-bit value transmitted (after Hamming encoding) was 0xA4F. The original 8-bit data value was 0xAF.

Problem

- What is the remainder obtained by dividing x^7+x^5+1 by the generator polynomial x^3+1?

Problem

- What is the remainder obtained by dividing x^7+x^5+1 by the generator polynomial x^3+1?

Answer:

- The remainder is x^2+x+1.

Problem

- Given the dataword 101001111 and the divisor 10111. Show the generation of the CRC codeword at the sender site (using binary division).

Problem

- Given the dataword 101001111 and the divisor 10111. Show the generation of the CRC codeword at the sender site (using binary division).

Answer:

The codeword at the sender site is 1010011110101