COMPUTER NETWORKS

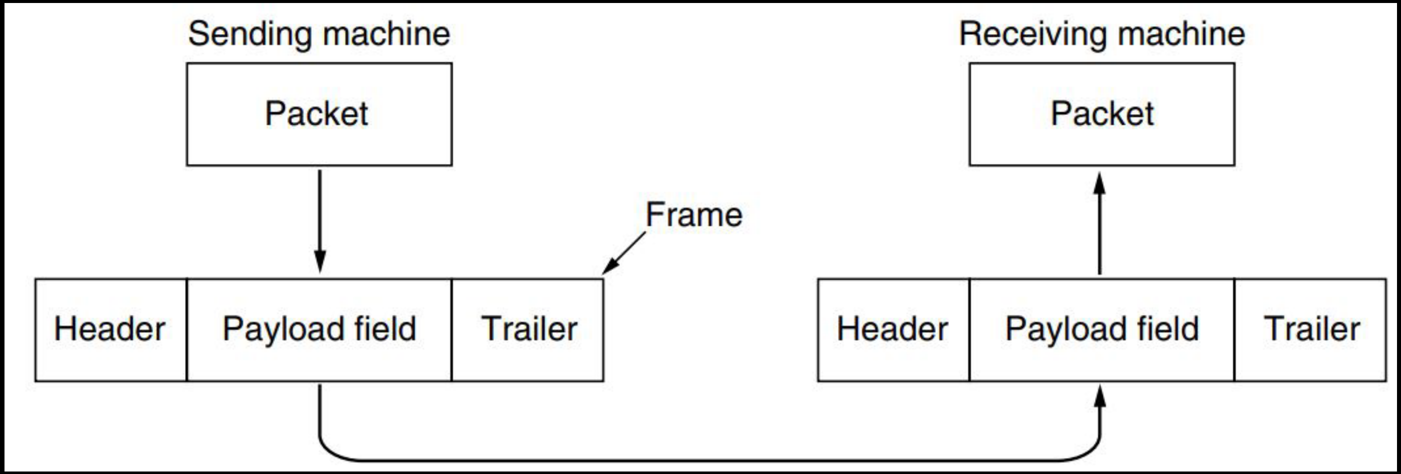
- Design issues
  - Framing
  - Error detection and correction
- Elementary data link protocols
  - Simplex protocol
  - A simplex stop and wait protocol for an error-free channel
  - A simplex stop and wait protocol for noisy channel
- Sliding Window protocols
  - A one-bit sliding window protocol
  - A protocol using Go-Back-N
  - A protocol using Selective Repeat
- Example data link protocols

Data Link Layer Design Issues

- 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:
  1. Providing a well-defined service interface to the network layer.
  2. Dealing with transmission errors.
  3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

Frame Management

- To accomplish these goals,
  - 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.



Relationship between packets and frames

- Error control and flow control, are found in transport and other protocols as well.
- That is because reliability is an overall goal, and it is achieved when all the layers work together.

Framing

- To provide service to the network layer, the data link layer must use the service provided to it by the physical layer
- The 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.

How Framing works

- When a frame is to be transmitted, the source machine does the following
  - 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.

How Framing works

- When a frame arrives at the destination, the receiver machine does the following
  - 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 (e.g., discarding the bad frame and possibly also sending back an error report)

The Four Framing Methods

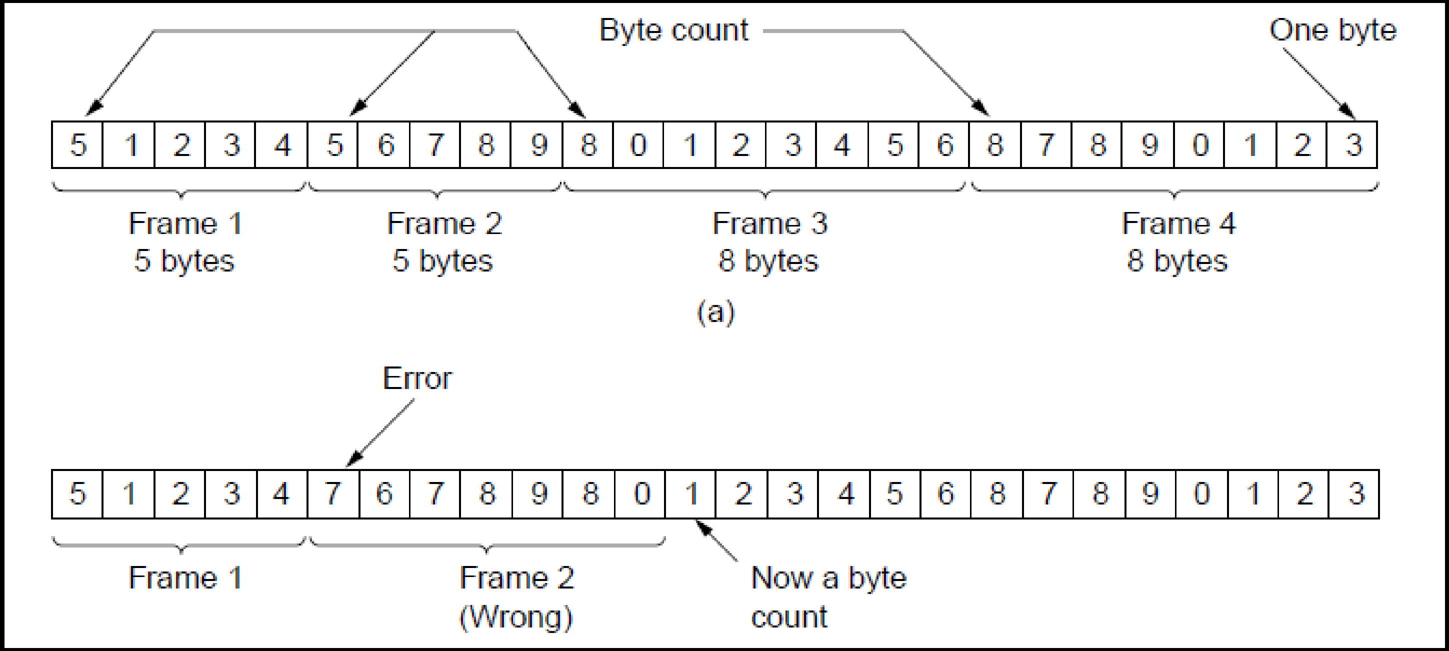
1. Byte count.
2. Flag bytes with byte stuffing.
3. Flag bits with bit stuffing.
4. Physical layer coding violations.

Framing using Byte count

- Byte count: specify the number of bytes in the frame
  - This framing method uses a field in the header to specify the number of bytes in the frame.
  - When the data link layer at the destination sees the byte count, it knows how many bytes follow and hence where the end of the frame is.
- Drawback:
  - The trouble with this algorithm is that the count can be garbled by a transmission error.
  - It will then be difficult to locate the correct start of the next frame.

Framing Byte Count

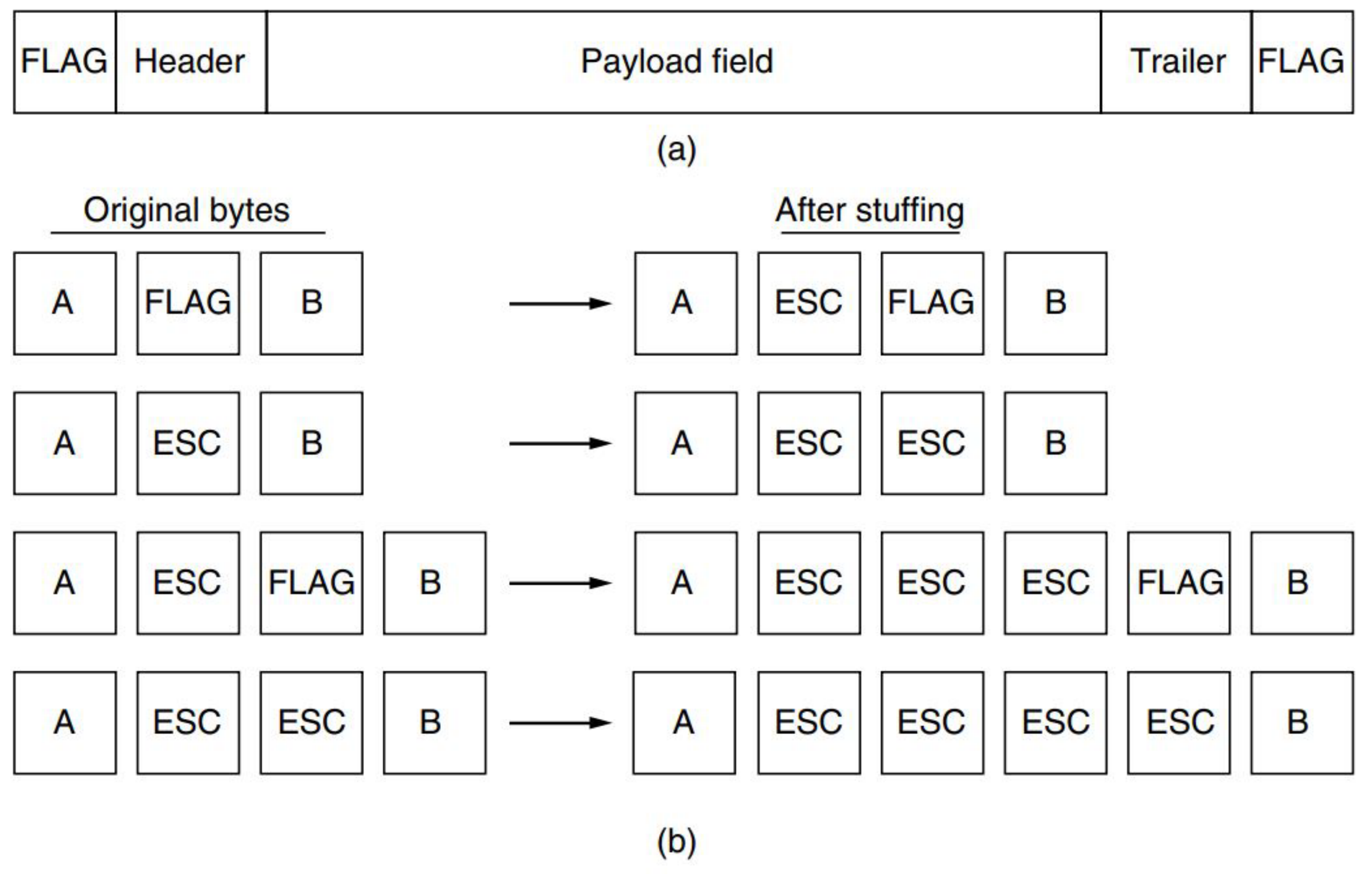
- Frame begins with a count of the number of bytes in it
  - Simple, but difficult to resynchronize after an error



Framing using byte stuffing

- Flag bytes with byte stuffing: Flag byte is used as both the starting and ending delimiter
  - The second framing 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.
  - Two consecutive flag bytes indicate the end of one frame and the start of the next.
  - To differentiate between byte pattern and data, a special ESC byte is inserted.

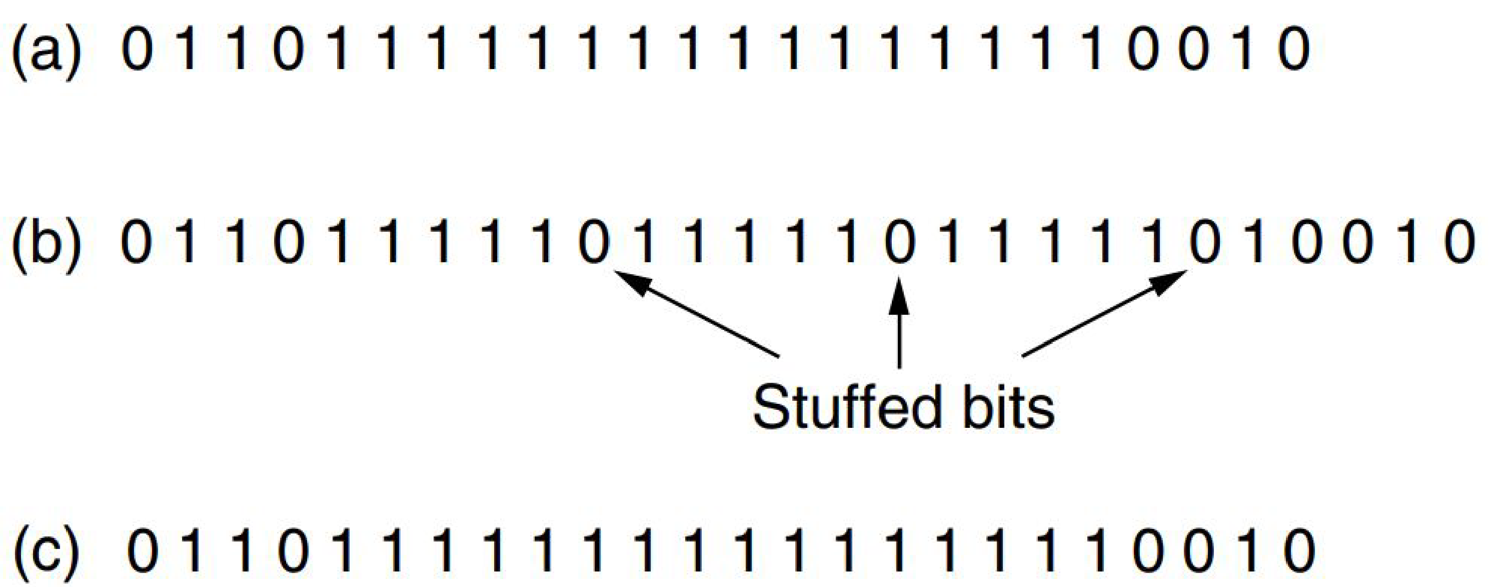
Framing using byte stuffing



Framing using bit stuffing

- Flag bits with bit stuffing: Each frame begins and ends with a special bit pattern (ex:01111110)
  - The third method of delimiting the bit stream gets around a disadvantage of byte stuffing, which is that it is tied to the use of 8-bit bytes.
  - Each frame begins and ends with a special bit pattern, 01111110
  - It also ensures a minimum density of transitions that help the physical layer maintain synchronization

Framing using bit stuffing



(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

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 de-stuffing.

Framing using reserved signals

- Physical layer coding violations: use some reserved signals to indicate the start and end of frames
  - The signal levels that are unused during line encoding schemes are utilized for highlighting frame delimitation.
- Generally, data link protocols use a combination of these methods for safety

Problems

Q) 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 frame:

A B ESC FLAG

when each of the following framing methods is used:

(a) Byte count.

(b) Flag bytes with byte stuffing.

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

Q) The following data fragment occurs in the middle of a data stream for which the byte stuffing algorithm described in the text is used:

A B ESC C ESC FLAG FLAG D.

What is the output after stuffing?

Q) A bit string, 0111101111101111110, needs to be transmitted at the data link layer.

What is the string actually transmitted after bit stuffing?

Q) What is the maximum overhead in byte-stuffing algorithm?

Error Detection and Correction

- Two basic strategies for dealing with errors
  1. Using error-correcting codes or FEC (Forward Error Correction) :
     - Include enough redundant information to enable the receiver to deduce what the transmitted data must have been.
  2. Using error-detecting codes or BEC (Backward Error Correction) :
     - Include only enough redundancy to allow the receiver to deduce that an error has occurred (but not which error) and have it request a retransmission.

Error Correcting codes

- □ Error-correcting codes are widely used on wireless links, which are notoriously noisy and error prone when compared to optical fibers. Examples include

1. Hamming codes.
2. Binary convolutional codes.
3. Reed-Solomon codes.
4. Low-Density Parity Check codes.

Error Correcting codes

- ECCs can be broadly categorized into two types –
  - **Block codes** – The message is divided into fixed-sized blocks of bits, to which redundant bits are added for error detection or correction.
  - **Convolutional codes** – The message comprises of data streams of arbitrary length and parity symbols are generated by the sliding application of a Boolean function to the data stream.

Hamming Distance

- The number of bit positions in which two codewords differ is called the Hamming distance
- As a simple example of an error-correcting code, consider a code with only four valid codewords:
  - 0000000000,
  - 0000011111,
  - 1111100000, and
  - 1111111111
- This code has a distance of 5.
- If the codeword 0000000111 arrives and we expect only single- or double-bit errors, the receiver will know that the original must have been 0000011111.

Hamming Code

- This technique is capable of detecting up to two simultaneous bit errors and correcting single-bit errors.
- Hamming code procedure used by the sender

  Step 1 − Calculation of the number of redundant bits.

  Step 2 − Positioning the redundant bits.

  Step 3 − Calculating the values of each redundant bit.

- Hamming code procedure used by the sender

  Step 1 − Calculation of the number of redundant bits.

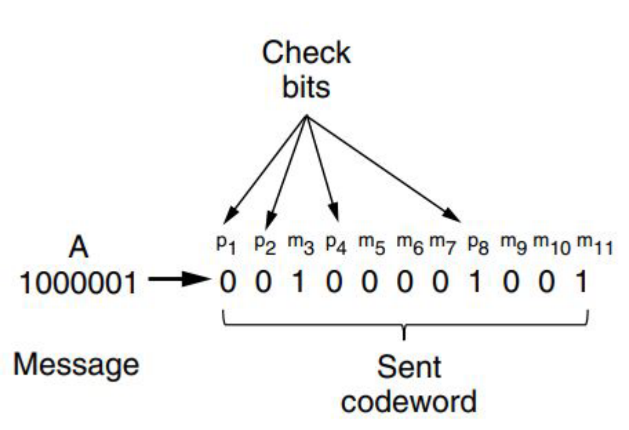
  Step 2 − Positioning the redundant bits.

  Step 3 − Parity checking.

  Step 4 − Error detection and correction

Hamming Code

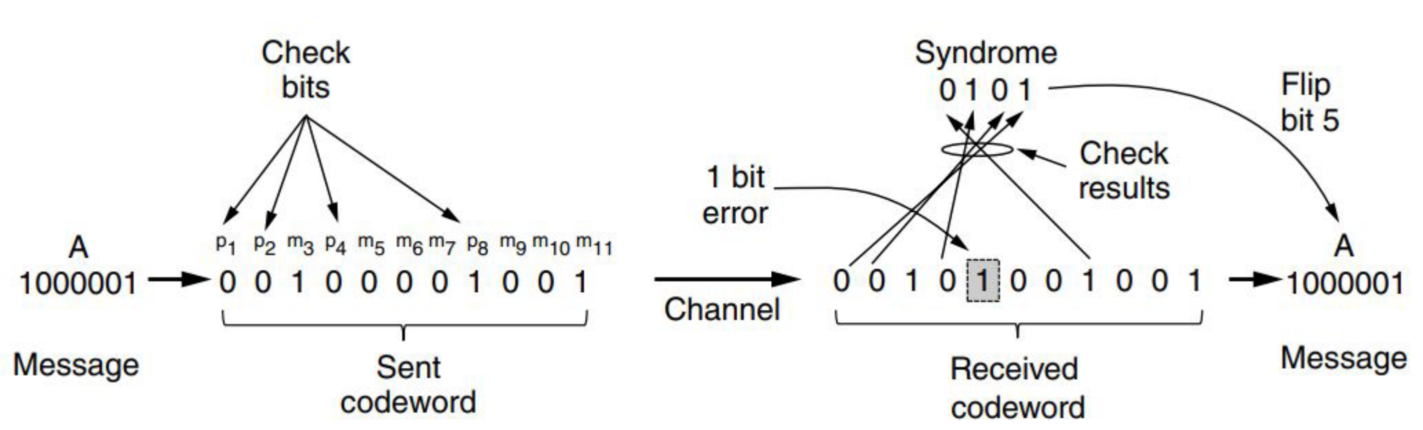
- Here the bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on.
- The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits.
- The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits.
- The check bits are computed for even parity sums for 1000001
- For example p1=p1,m3,m5,m7,m9,m11
- p1=p1,1,0,0,0,1
- Using even parity p1 gets a value of 0 (total number of 1’s is even)


Check bits
A
1000001
p_1 p_2 m_3 p_4 m_5 m_6 m_7 p_8 m_9 m_{10} m_{11}
0 0 1 0 0 0 0 1 0 0 1
Message
Sent codeword

Hamming Code

- Original data: 00100001001
- Original check results are 0(p1), 0(p2), 0(p4), and 1(p8)

- A single-bit error occurred on the channel,
- The check results are 0(p1), 1(p2), 0(p4), and 1(p8).
- This gives a value of 0101 or 4 + 1=5.
- By the design of the scheme, this means that the fifth bit is in error.

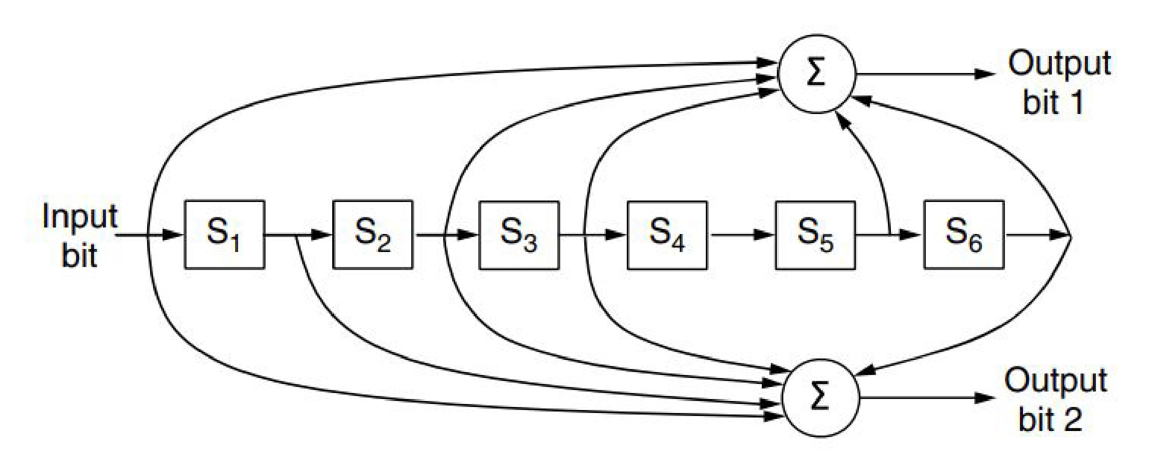


Sixteen-bit messages are transmitted using a Hamming code. How many check bits are needed to ensure that the receiver can detect and correct single-bit errors? Show the bit pattern transmitted for the message 1101001100110101. Assume that even parity is used in the Hamming code.

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.

Binary convolutional codes

- In a convolutional code, an encoder processes a sequence of input bits and generates a sequence of output bits.
- The output depends on the current and previous input bits.
- Each input bit on the left-hand side produces two output bits on the right-hand side that are XOR sums of the input and internal state.



Reed-Solomon codes

- Reed-Solomon codes are based on the fact that every n degree polynomial is uniquely determined by n + 1 points.
- For example, a line having the form ax + b is determined by two points.
- Imagine that we have two data points that represent a line and we send those two data points plus two check points chosen to lie on the same line.
- If one of the points is received in error, we can still recover the data points by fitting a line to the received points.
- Three of the points will lie on the line, and one point, the one in error, will not.
- By finding the line we can corrected the error.

Low-Density Parity Check codes

- Each output bit is formed from only a fraction of the input bits.
- This leads to a matrix representation of the code that has a low density of 1s, hence the name for the code.
- The received codewords are decoded with an approximation algorithm that iteratively improves on a best fit of the received data to a legal codeword.
- This corrects errors.

Error-Detecting Codes

- Over optical fiber or high-quality copper, the error rate is much lower.
- Error detection and retransmission is usually more efficient for dealing with the occasional error. Examples include

1. Parity.
2. Checksums.
3. Cyclic Redundancy Checks (CRCs).

Parity

- Here a single bit is appended to the data.
- The bit is chosen so that the number of 1 bits in the codeword is even (or odd).
- Doing this is equivalent to computing the (even) parity bit as the modulo 2 sum or XOR of the data bits.

- For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100.
- With odd parity 1011010 becomes 10110101.
- A code with a single parity bit has a distance of 2, since any single-bit error produces a codeword with the wrong parity
- This means that it can detect single-bit errors but no double bit errors.

Checksums

- Used in Internet protocols (IP, ICMP, TCP, UDP)
  - Basic Idea: Add up the data and send it along with sum
- The word "checksum" is often used to mean a group of check bits associated with a message, regardless of how are calculated.
- The checksum is usually placed at the end of the message, as the complement of the sum function.
- This way, errors may be detected by summing the entire received codeword, both data bits and checksum.
- If the result comes out to be zero, no error has been detected.

CRC (Cyclic Redundancy Check)

- Stronger protection than checksums
- Used widely in practice, e.g., Ethernet/802.11 CRC-32
- Algorithm: Given n bits of data, generate a k bit check sequence that gives a combined n + k bits that are divisible by a chosen divisor C(x) a chosen divisor C(x)
- Based on mathematics of finite fields – "numbers" correspond to polynomials use modulo arithmetic "numbers" correspond to polynomials, use modulo arithmetic
  - e.g. interpret 10011010 as x^7 + x^4 + x^3 + x^1

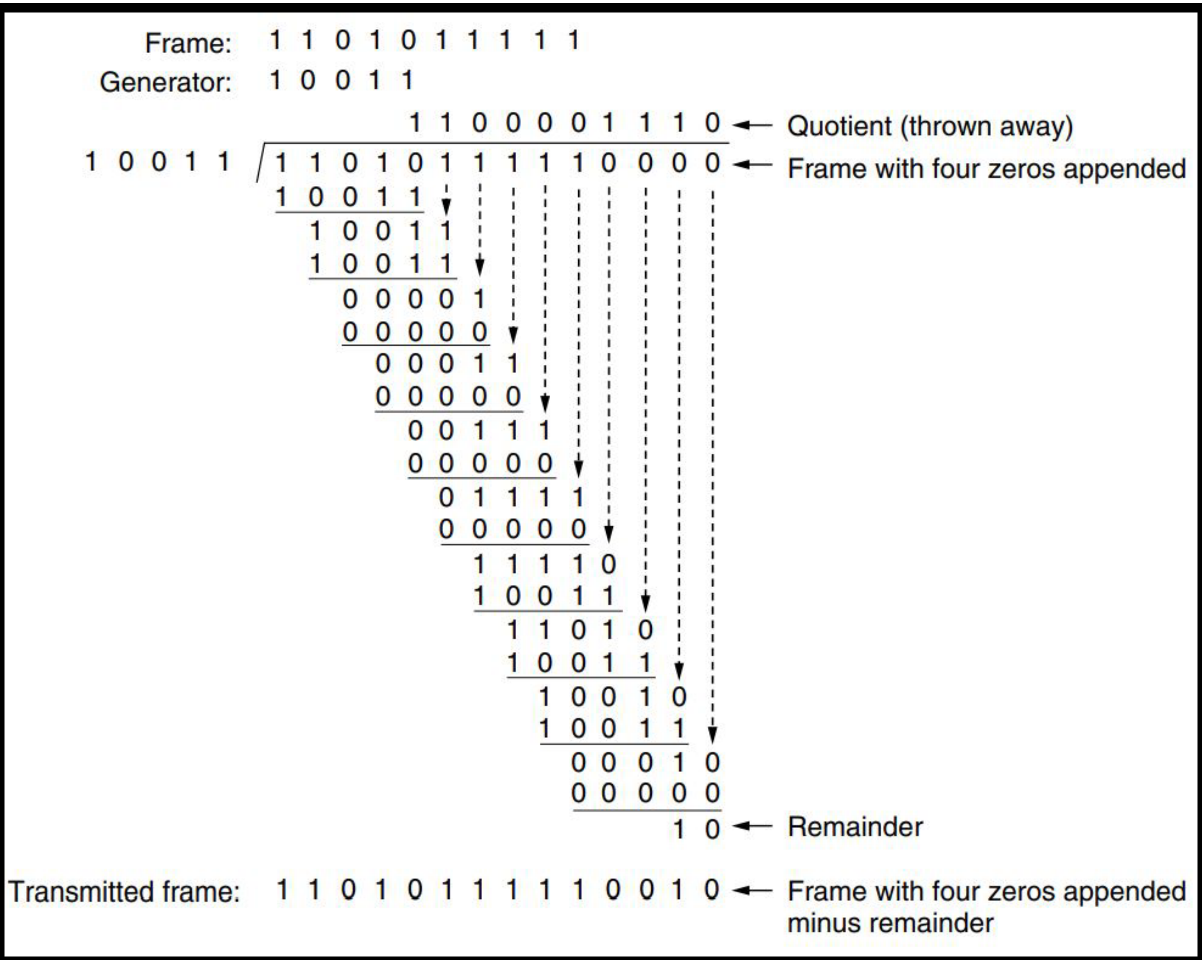
CRC (Cyclic Redundancy Check)

- **Algorithm for Encoding using CRC**
  - The communicating parties agrees upon the size of message, *M(x)* and the generator polynomial, *G(x)*.
  - If *r* is the order of *G(x)*, *r* bits are appended to the low order end of *M(x)*. This makes the block size bits, the value of which is $x^rM(x)$.
  - The block $x^rM(x)$ is divided by *G(x)* using modulo 2 division.
  - The remainder after division is added to $x^rM(x)$ using modulo 2 addition. The result is the frame to be transmitted, *T(x)*. The encoding procedure makes exactly divisible by *G(x)*.

CRC (Cyclic Redundancy Check)

- **Algorithm for Decoding using CRC**
  - The receiver divides the incoming data frame T(x) unit by G(x) using modulo 2 division. Mathematically, if E(x) is the error, then modulo 2 division of [$M(x) + E(x)$] by $G(x)$ is done.
  - If there is no remainder, then it implies that $E(x)$. The data frame is accepted.
  - A remainder indicates a non-zero value of E(x), or in other words presence of an error. So the data frame is rejected.
  - The receiver may then send an erroneous acknowledgment back to the sender for retransmission.

Example of CRC calculation



Frame: 1 1 0 1 0 1 1 1 1 1

Generator: 1 0 0 1 1

```
                    1 1 0 0 0 0 1 1 1 0   ← Quotient (thrown away)
1 0 0 1 1 / 1 1 0 1 0 1 1 1 1 1 0 0 0 0   ← Frame with four zeros appended
            1 0 0 1 1
            ---------
              1 0 0 1 1
              1 0 0 1 1
              ---------
                0 0 0 0 1
                0 0 0 0 0
                ---------
                  0 0 0 1 1
                  0 0 0 0 0
                  ---------
                    0 0 1 1 1
                    0 0 0 0 0
                    ---------
                      0 1 1 1 1
                      0 0 0 0 0
                      ---------
                        1 1 1 1 0
                        1 0 0 1 1
                        ---------
                          1 1 0 1 0
                          1 0 0 1 1
                          ---------
                            1 0 0 1 0
                            1 0 0 1 1
                            ---------
                              0 0 0 1 0
                              0 0 0 0 0
                              ---------
                                    1 0   ← Remainder
```

Transmitted frame: 1 1 0 1 0 1 1 1 1 1 0 0 1 0 ← Frame with four zeros appended minus remainder

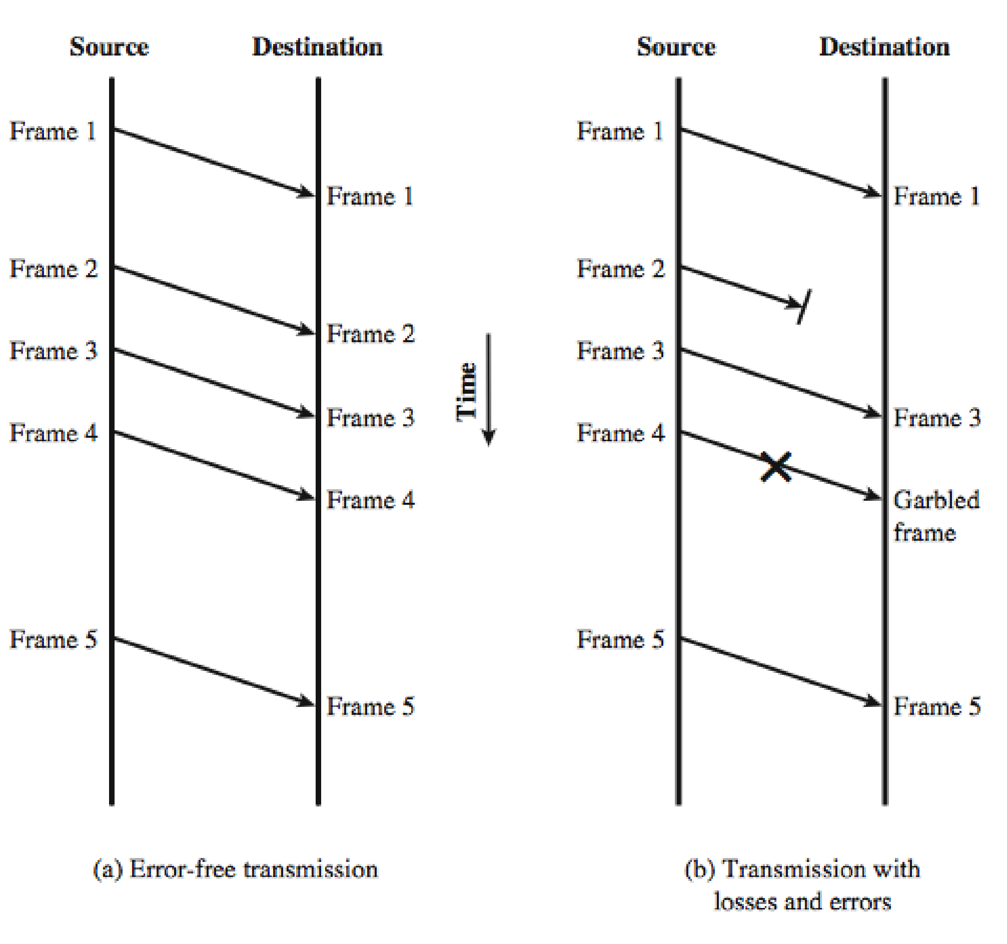
Real Error detection/ Correction codes

- Detection (often at link/network/transport layers)
  - Parity, simple example
  - Checksums, but weak
  - CRCs, widely used
- Correction (often at physical and application layers)
  - Hamming codes, simple example
  - Convolutional codes
  - Reed-Solomon
  - Low-density Parity Check (LDPC) codes

Elementary Data Link Protocols

- We begin with three simple but unrealistic protocols.
  - An unrestricted simplex protocol
  - A simplex stop-and-wait protocol
  - A simplex protocol for a noisy channel

Frame Transmissions Types

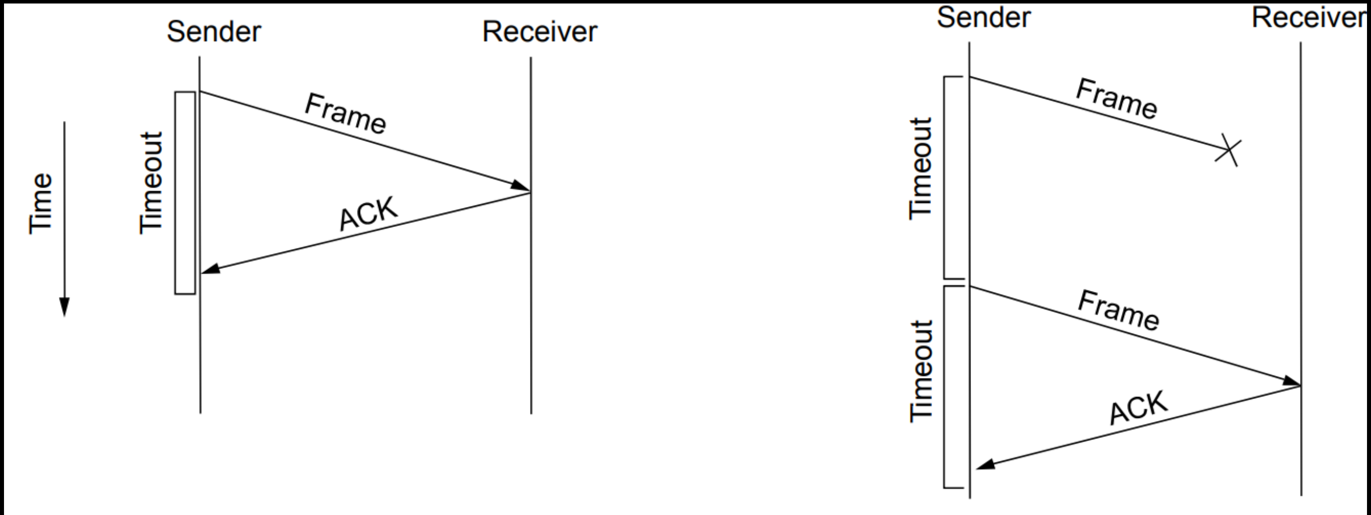


(a) Error-free transmission

(b) Transmission with losses and errors

ARQ (Automatic Repeat Request)

- Packets can be corrupted or lost.
  - How do we add reliability?
    - Acknowledgments (ACKs) and retransmissions after a timeout
    - Automatically resend until a positive acknowledgement is received
    - ARQ is generic name for protocols based on this strategy

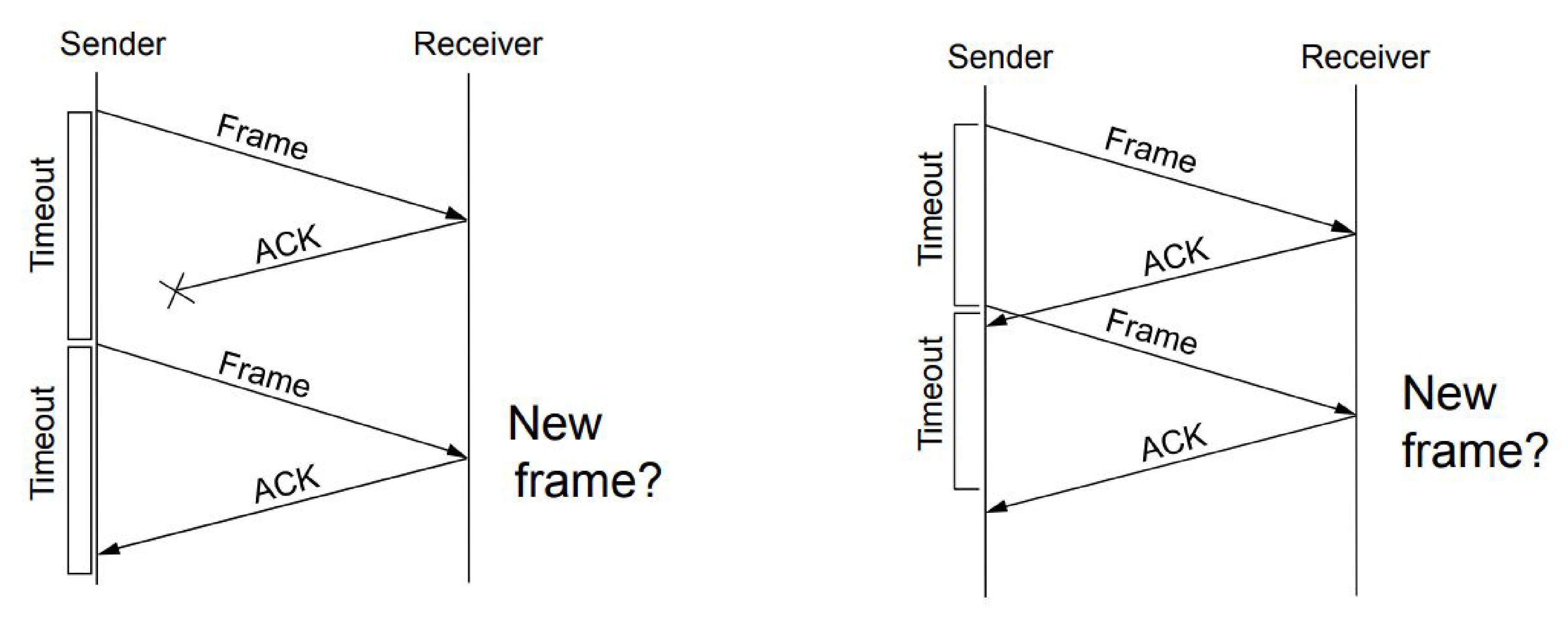


Two Issues

- How long to set the timeout?
  - Only easy on a direct link, otherwise timing variability
  - Way too long lowers performance
  - Implies sometimes timeout will be early

- How to avoid accepting duplicate frames as new
  - Given retransmissions, frame loss and imprecise timeouts
  - Answer: Sequence numbers

The Need for Sequence Numbers



```
#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 frame is it? */
  seq_nr seq;                                 /* sequence number */
  seq_nr ack;                                 /* acknowledgement number */
  packet info;                                /* the network layer packet */
} frame;


/* 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);
```

Some definitions needed in the protocols to follow. These definitions are located in the file protocol.h.

```
/* 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
```

An unrestricted simplex protocol

- We will begin with a simple but unrealistic protocol.
- In this protocol:
  - Data are transmitted in one direction only
  - The transmitting (Tx) and receiving (Rx) hosts are always ready
  - Processing time can be ignored
  - Infinite buffer space is available
  - No errors occur; i.e. no damaged frames and no lost frames

```
Sender()
{
        forever
       {
             from_host(buffer);
              S.info = buffer;
              sendframe(S);
        }
 }
```

```
 Receiver()
 {
          forever
          {
                wait(event);
getframe(R);
                to_host(R.info);
           }
 }
```

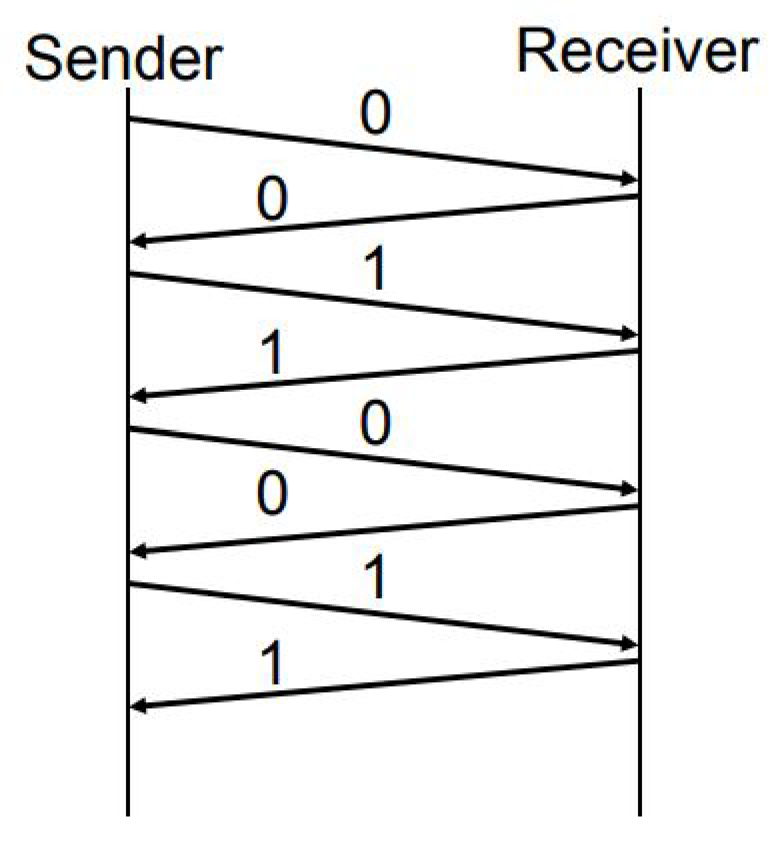
```
/* Protocol 1 (Utopia) provides for data transmission in one direction only, from   sender to receiver. The communication
channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.  Consequently, the sender just sits in a
loop pumping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

A simplex stop-and-wait protocol

- In this protocol we assume that
  - Data are transmitted in one direction only
  - No errors occur (perfect channel)
  - The receiver can only process the received information at a finite rate
- The problem here is how to prevent the sender from flooding the receiver.
- A general solution could be as follows:
  - The receiver send an acknowledge frame back to the sender.
  - The sender, after having sent a frame, must wait for the acknowledge frame from the receiver before sending another frame.

1 Bit Stop and Wait



- Only one outstanding frame at a time, 0 or 1
- Retransmissions re-sent with same number
- Number only needs to distinguish between current and next frame
  - A single bit will do
- Want to utilize all available bandwidth
  - Need to keep more data "in flight"
  - How much?
    - Depends on the bandwidth-delay product?
- **Leads to Sliding Window Protocol**

A simplex stop-and-wait protocol

```
Sender()
{
        forever
        {
          from_host(buffer);
          S.info = buffer;
          sendframe(S);
          wait(event);
        }
}
```

```
Receiver()
{
      forever
      {
           wait(event);
           getframe(R);
           to_host(R.info);
            sendframe(S);
       }
}
```

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

A simplex protocol for noisy channel

- In this protocol the unreal "error free" assumption in protocol 2 is dropped.
- Frames may be either damaged or lost completely.
- We assume that transmission errors in the frame are detected by the hardware checksum.
- The receiver needs to distinguish only 2 possibilities:
  - A new frame or a duplicate; a 1-bit sequence number is sufficient.
  - At any instant the receiver expects a particular sequence number.
  - Any wrong sequence numbered frame arriving at the receiver is rejected as a duplicate.
  - A correctly numbered frame arriving at the receiver is accepted, passed to the host, and the expected sequence number is incremented by 1 (modulo 2).

A simplex protocol for noisy channel

```
Sender()
{
    NFTS = 0;  /* NFTS = Next Frame To Send */
    from_host(buffer);
    forever
    {
        S.seq = NFTS;
        S.info = buffer;
        sendf(S);
        start_timer(S.seq);
        wait(event);
        if(event == frame_arrival)
        {
            from_host(buffer);
            ++NFTS;  /* modulo 2 operation */
        }
    }
}
```

```
Receiver()
{
    FE = 0;                /* FE = Frame Expected */
    forever
    {
        wait(event);
        if(event == frame_arrival)
        {
            getf(R);
            if(R.seq == FE)
            {
                 to_host(R.info);
                ++FE;  /* modulo 2 operation */
            }
            sendf(S);      /* ACK */
        }
    }
}
```

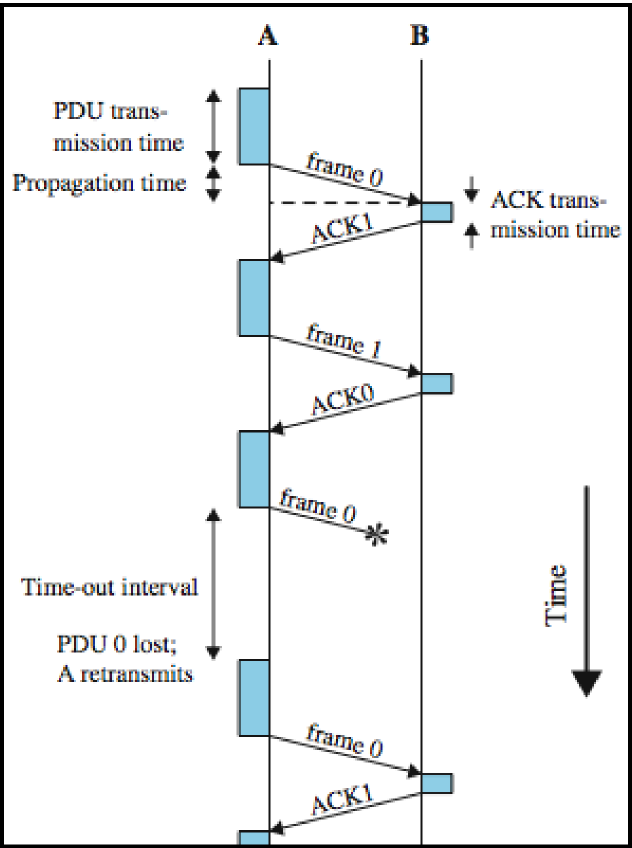
```
/* Protocol 3 (PAR) allows unidirectional data flow over an unreliable channel. */ PROTOCOL FOR NOISY CHANNEL
#define MAX_SEQ 1                     /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
```

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

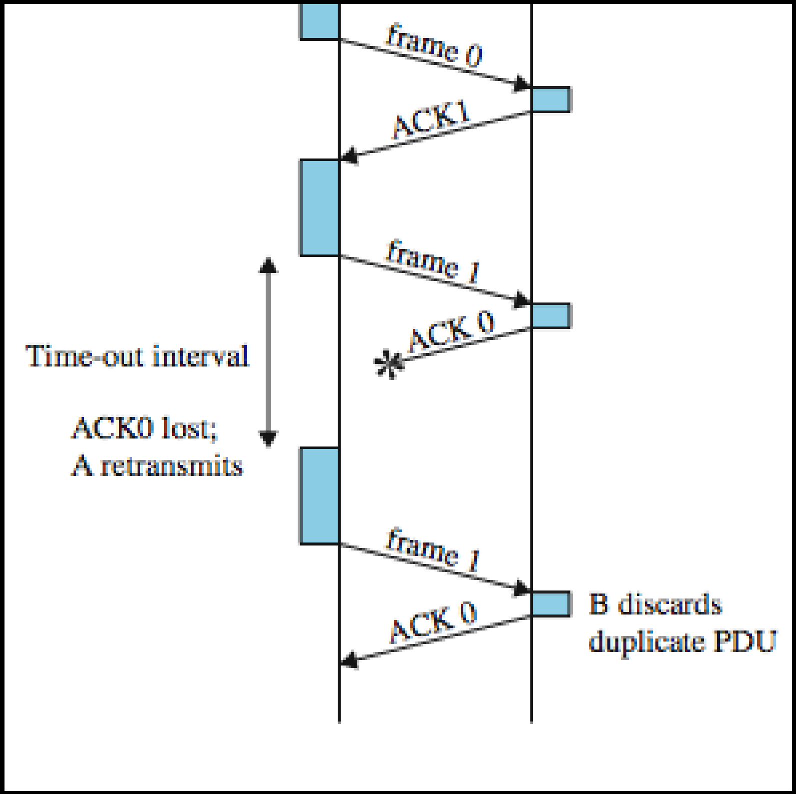
```
void receiver3(void)
{
  seq_nr frame_expected;
```

```
   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 */
        }
  }
}
```

Timeout and Retransmissions



Timeout and frame retransmission



ACK lost and frame retransmission

Sliding Window Protocols

- In most practical situations there is a need for transmitting data in both directions (i.e. between 2 computers).
- If protocol 2 or 3 is used in these situations the data frames and ACK (control) frames in the reverse direction have to be interleaved.
- This method is acceptable but not efficient.
- An efficient method is to absorb the ACK frame into the header of the data frame going in the same direction.
- **This technique is known as *piggybacking*.**
- If a new host packet arrives quickly the acknowledgement is piggybacked onto it; otherwise, the host just sends a separate ACK frame.

Sliding Window Protocols

- When one host sends traffic to another it is desirable that the traffic should arrive in the same *sequence* as that in which it is dispatched.
- It is also desirable that a data link should deliver frames in the order sent.
- A flexible concept of sequencing is referred to as the *sliding window* concept.
- In all sliding window protocols, each outgoing frame contains a sequence number SN ranging from 0 to 2^(n -1) where n is the number of bits reserved for the sequence number field.
- At any instant of time the sender maintains a list of consecutive sequence numbers corresponding to frames it is permitted to send.
- These frames are said to fall within the sending window.

Sliding Window Protocols

- Similarly, the receiver maintains a receiving window corresponding to frames it is permitted to accept.
- The size of the window relates to the available buffers of a receiving or sending node at which frames may be arranged into sequence.
- At the receiving node, any frame falling outside the window is discarded.
- Frames falling within the receiving window are accepted and arranged into sequence.
- Once sequenced, the frames at the left of the window are delivered to the host and an acknowledgement of the delivered frames is transmitted to their sender.

Sliding Window Protocols

- The window is then rotated to the position where the left edge corresponds to the next expected frame, RN.
- Whenever a new frame arrives from the host, it is given the next highest sequence number, and the upper edge of the sending window is advanced by one.
- The sequence numbers within the sender's window represent frames sent but as yet not acknowledged.
- When an acknowledgement comes in, it gives the position of the receiving left window edge which indicates what frame the receiver expects to receive next.
- The sender then rotates its window to this position, thus making buffers available for continuous transmission.

Sliding Window concept

Sender maintains window of frames it can send

- Needs to buffer them for possible retransmission
- Window advances with next acknowledgements

Receiver maintains window of frames it can receive

- Needs to keep buffer space for arrivals
- Window advances with in-order arrivals

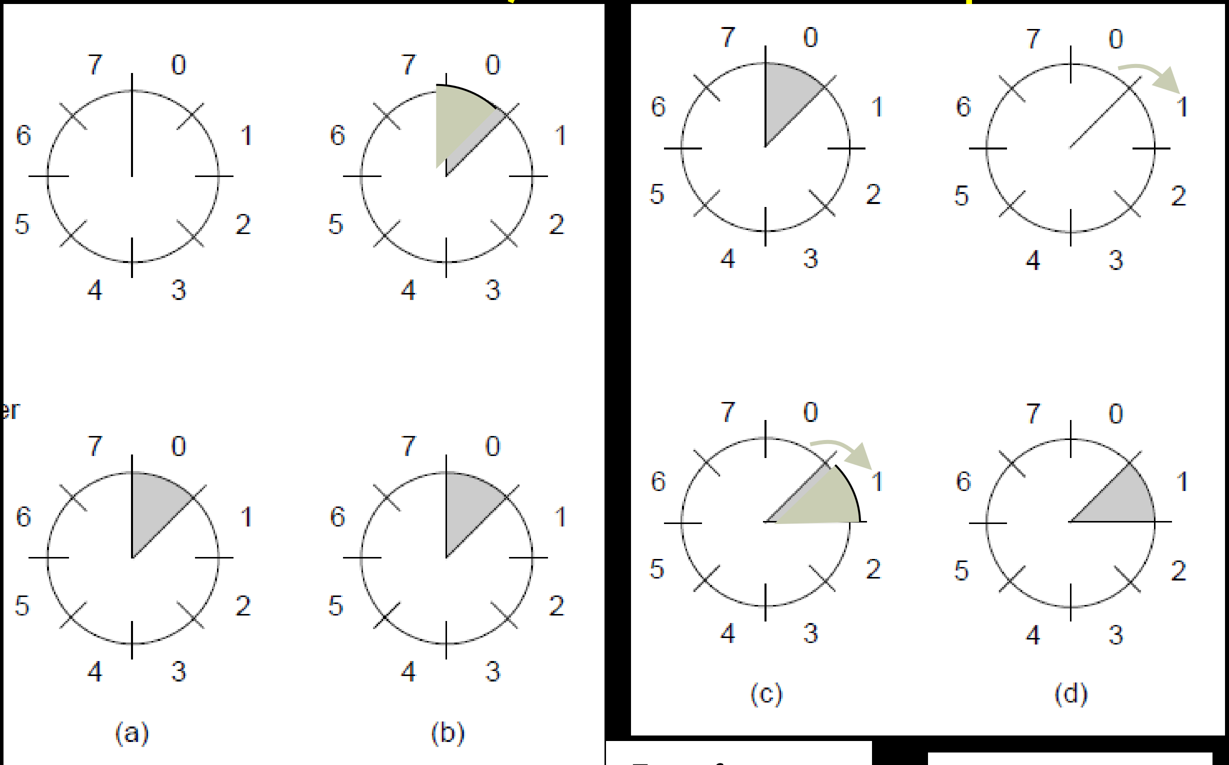
Sliding Window Protocols

- Types
  - A one-bit sliding window protocol
  - A protocol using Go-Back-N
  - A protocol using Selective Repeat

Sliding Window concept

A sliding window advancing at the sender and receiver

- Ex: window size is 1, with a 3-bit sequence number.



Sender

Receiver

At the start

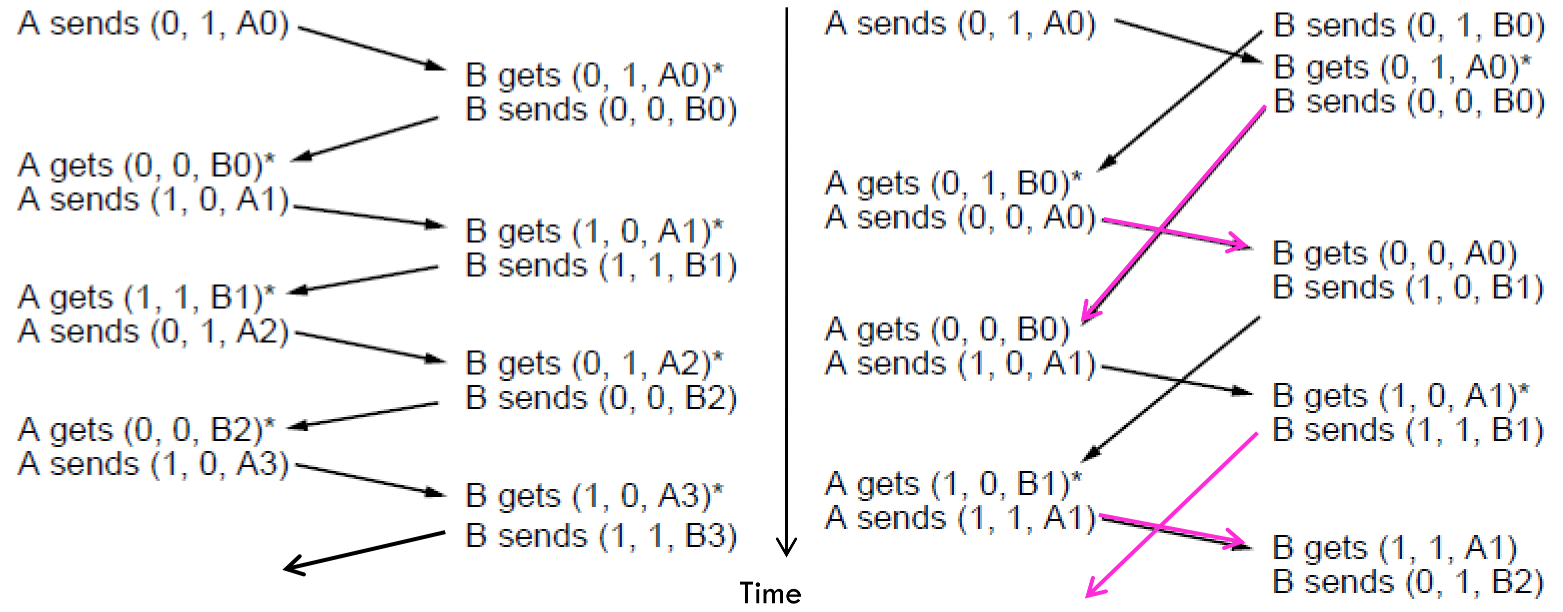
First frame is sent

First frame is received

Sender gets first ack

One-Bit Sliding Window

- Two scenarios show subtle interactions exist in p4:
  - Simultaneous start [right] causes correct but slow operation



A sends (0, 1, A0)
B gets (0, 1, A0)*
B sends (0, 0, B0)
A gets (0, 0, B0)*
A sends (1, 0, A1)
B gets (1, 0, A1)*
B sends (1, 1, B1)
A gets (1, 1, B1)*
A sends (0, 1, A2)
B gets (0, 1, A2)*
B sends (0, 0, B2)
A gets (0, 0, B2)*
A sends (1, 0, A3)
B gets (1, 0, A3)*
B sends (1, 1, B3)

Time

A sends (0, 1, A0)
B sends (0, 1, B0)
B gets (0, 1, A0)*
B sends (0, 0, B0)
A gets (0, 1, B0)*
A sends (0, 0, A0)
B gets (0, 0, A0)
B sends (1, 0, B1)
A gets (0, 0, B0)
A sends (1, 0, A1)
B gets (1, 0, A1)*
B sends (1, 1, B1)
A gets (1, 0, B1)*
A sends (1, 1, A1)
B gets (1, 1, A1)
B sends (0, 1, B2)

Notation is (seq, ack, frame number). Asterisk indicates frame accepted by network layer .

Normal case

Correct, but poor performance

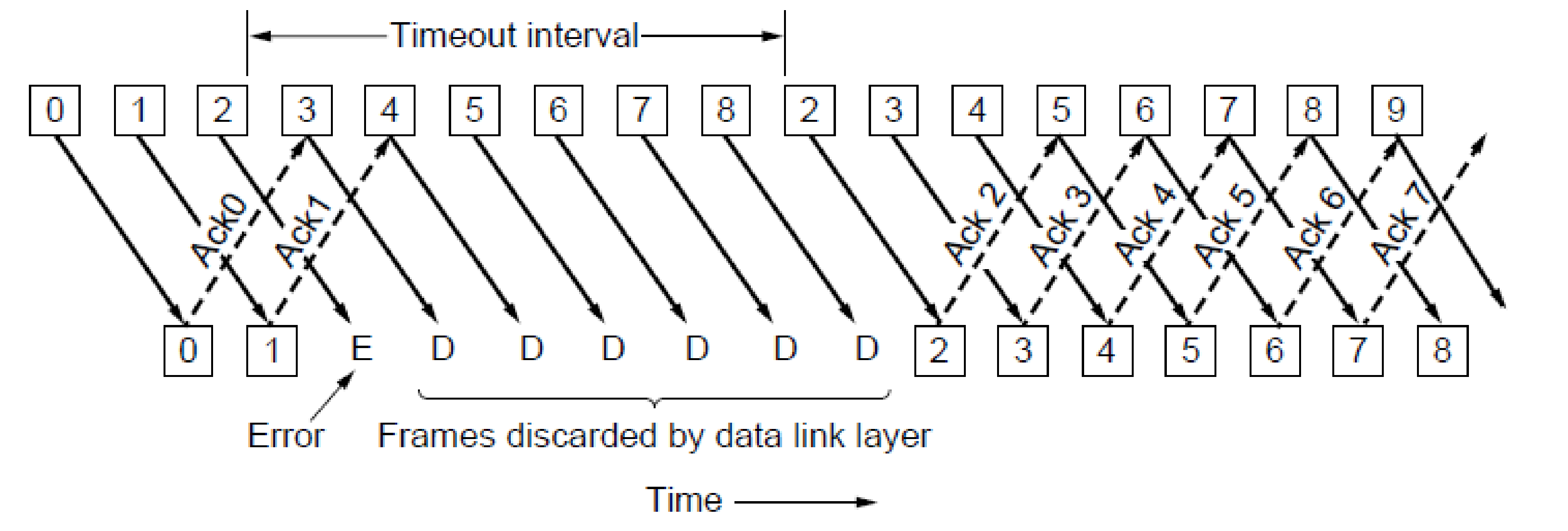
Sliding Window concept

- Larger windows enable pipelining for efficient link use
  - Stop-and-wait (w=1) is inefficient for long links
  - Best window (w) depends on bandwidth-delay (BD)
  - Want w ≥ 2BD+1 to ensure high link utilization
- Pipelining leads to different choices for errors/buffering
  - We will consider Go-Back-N and Selective Repeat

Go-Back-N

Receiver only accepts/acks frames that arrive in order:

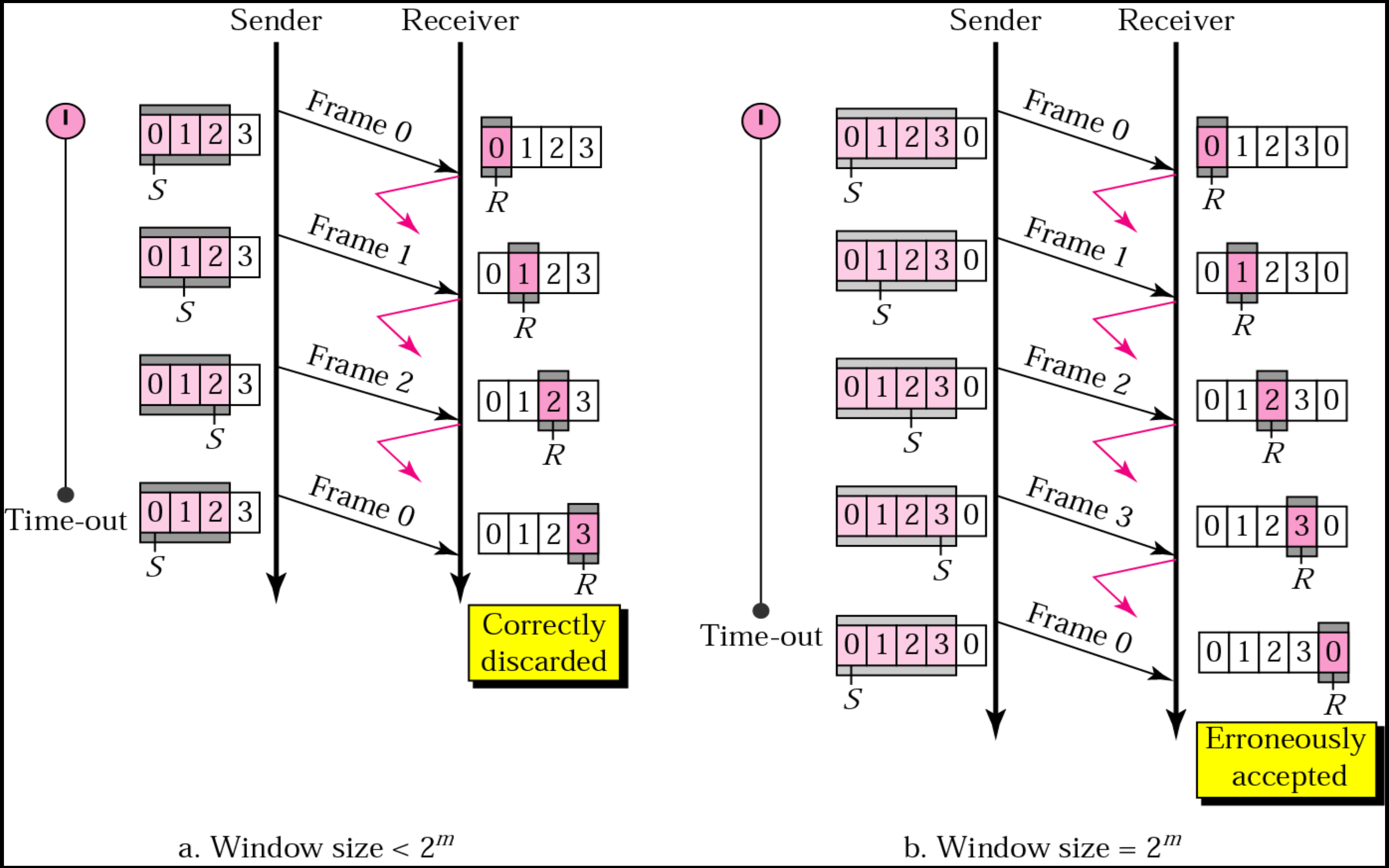
- Discards frames that follow a missing/errored frame
- Sender times out and resends all outstanding frames



Go-Back-N ARQ, sender window size

- Size of the sender window must be less than 2^m. Size of the receiver is always 1. If m = 2, window size = $2^m - 1 = 3$.

- Fig compares a window size of 3 and 4.



a. Window size < 2^m

b. Window size = 2^m

Accepts as the 1st frame in the next cycle-an error

Go-Back-N

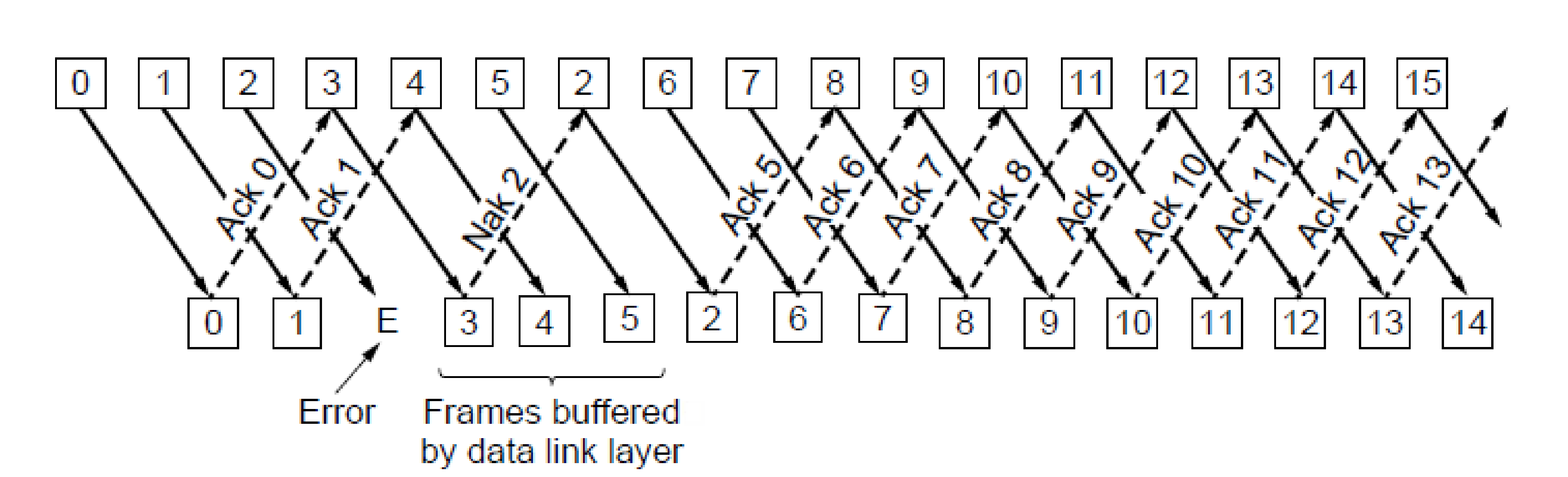
Tradeoff made for Go-Back-N:

- Simple strategy for receiver; needs only 1 frame
- Wastes link bandwidth for errors with large windows; entire window is retransmitted

Selective Repeat

Receiver accepts frames anywhere in receive window

- Cumulative ack indicates highest in-order frame
- NAK (negative ack) causes sender retransmission of a missing frame before a timeout resends window



Selective Repeat

Tradeoff made for Selective Repeat:

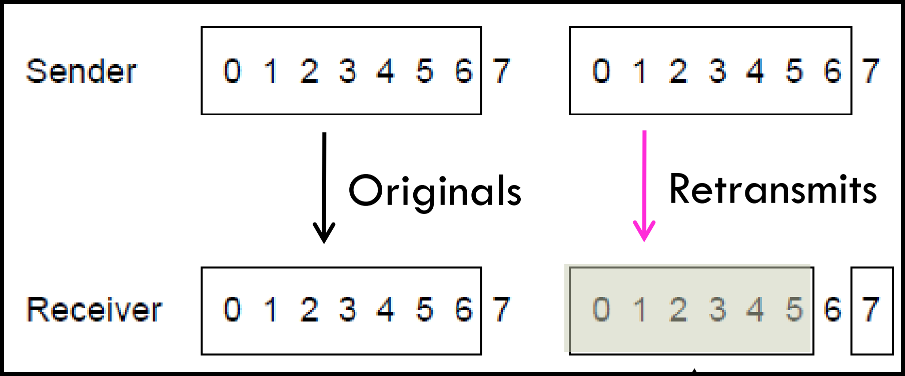
- More complex than Go-Back-N due to buffering at receiver and multiple timers at sender
- More efficient use of link bandwidth as only lost frames are resent (with low error rates)

Selective Repeat

For correctness, we require:

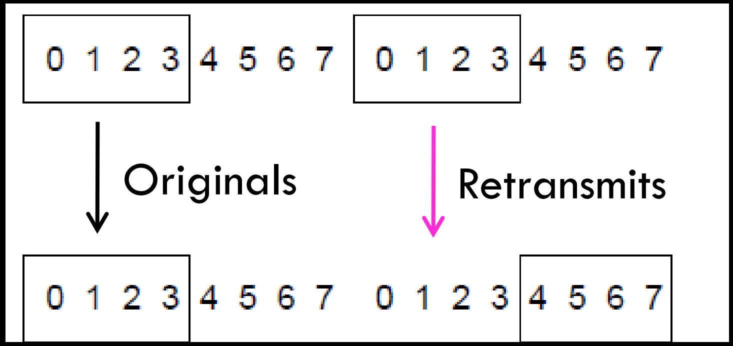
- Sequence numbers (s) at least twice the window (w)

Error case (s=8, w=7) – too few sequence numbers



New receive window overlaps old – retransmits ambiguous

Correct (s=8, w=4) – enough sequence numbers



New and old receive window don't overlap – no ambiguity

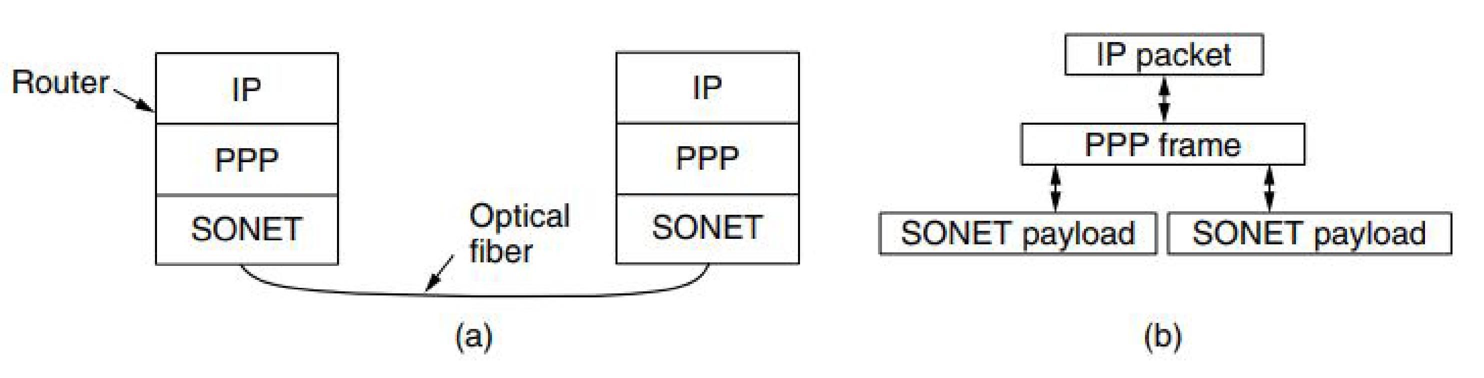
Example data link protocols

- For point-to-point links we have 2 examples
  1. The SONET optical fiber links are used in wide-area networks.
     - These links are widely used, for example, to connect routers in the different locations of an ISP’s network.
  2. The ADSL links run on the local loop of the telephone network at the edge of the Internet.
     - These links connect millions of individuals and businesses to the Internet.

SONET (Synchronous Optical Network)

1. The SONET optical fiber links are used in wide-area networks.

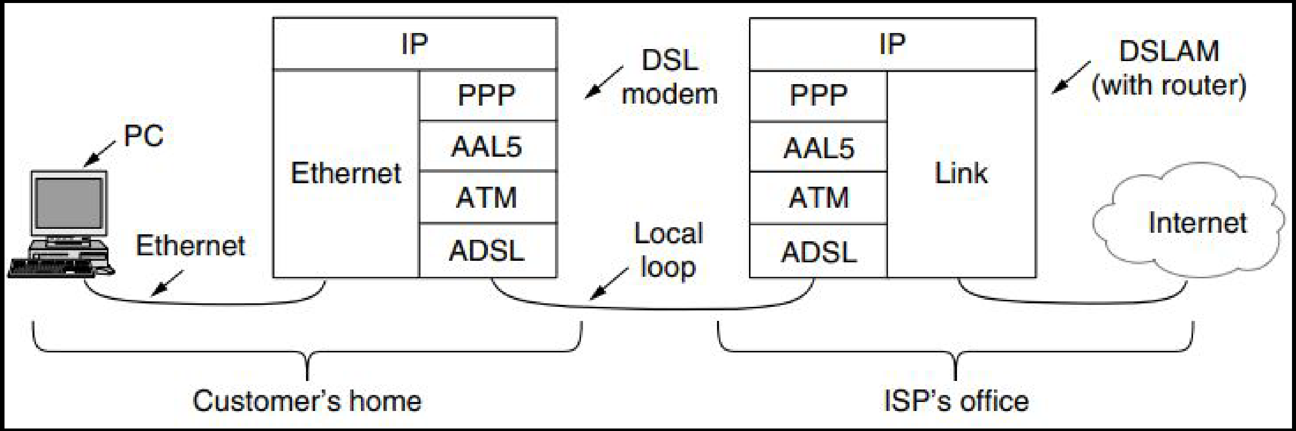
- □ These links are widely used, for example, to connect routers in the different locations of an ISP's network.
- □ Advantages of SONET:
  - □ Transmits data to large distances
  - □ Low electromagnetic interference
  - □ High data rates
  - □ Large Bandwidth



ADSL (Asymmetric Digital Subscriber Loop)

2. ADSL is a type of digital data transmission for Internet access over symmetric copper telephone line pairs.

- □ These links connect millions of individuals and businesses to the Internet.
- □ Advantages of ADSL:
  - □ It doesn't occupy the telephone line.
  - □ It employs traditional infrastructure.
  - □ Outperforms dial-up connection.
  - □ Allows central and customized circuits


IP
PPP
AAL5
ATM
ADSL
Ethernet
DSL modem
IP
PPP
AAL5
ATM
ADSL
Link
DSLAM (with router)
PC
Ethernet
Local loop
Internet
Customer's home
ISP's office