UNIT-III

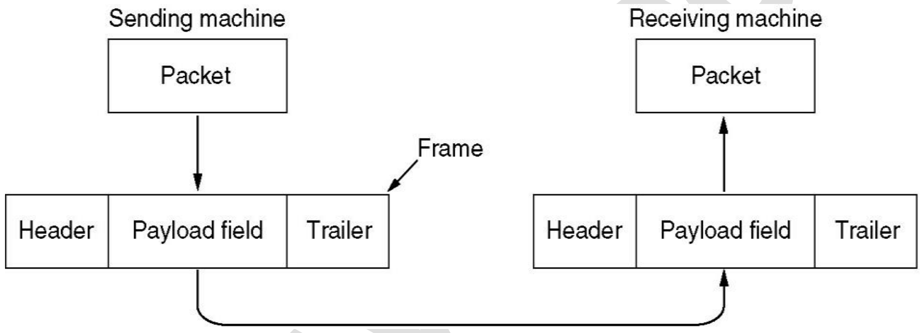
COMPUTER NETWORKS: DATALINK LAYER

Functions of datalink layer

- Services Provided to the Network Layer
- Framing
- Error Control
- Flow Control

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



- 'Physical addresses' are used in frame headers to identify source and destination

TYPES OF SERVICES PROVIDED TO THE NETWORK LAYER

- Unacknowledged Connectionless service
- Acknowledged Connectionless service
- Acknowledged Connection-Oriented service

Unacknowledged Connectionless service

- 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.

- This class of service is appropriate when the error rate is very low so that recovery is left to higher layers.
- It is also appropriate for real-time traffic, such as voice, in which late data are worse than bad data. Most LANs use unacknowledged connectionless service in the data link layer.

Acknowledged Connectionless service

No logical connections used,

Each frame sent is individually acknowledged. Sender knows whether a frame has arrived correctly. If it has not arrived within a specified time interval, it can be sent again.

This service is useful over unreliable channels, such as wireless systems.

Acknowledged Connection-Oriented service

When connection-oriented service is used, transfers go through three distinct phases. Connection is established, one or more frames are actually transmitted and then connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

FRAMING

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

Physical layer accept a raw bit stream and attempt to deliver it to the destination. This bit stream is not guaranteed to be error free. It is up to the data link layer to **detect and correct errors**.

To detect and correct errors, the usual approach used is to **break the bit stream up into discrete frames and compute the checksum** for each frame.

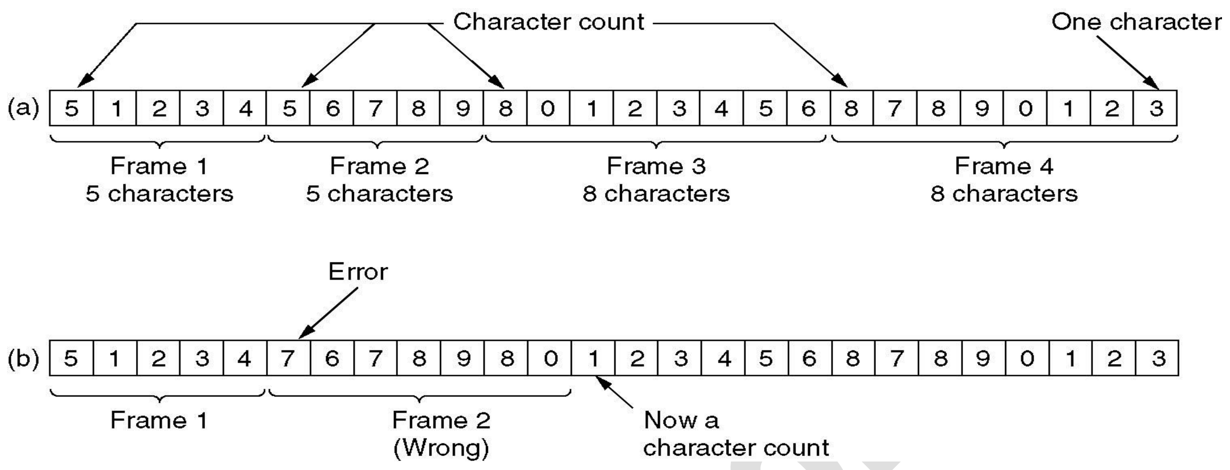
When a frame arrives at the destination, the checksum is recomputed. If the newly-computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it.

Four popular methods of framing are,

1. Character count.
2. Flag bytes with byte stuffing.
3. Starting and ending flags, with bit stuffing.

Character Count

It uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is. This technique is shown for four frames of sizes 5, 5, 8, and 8 characters, respectively.



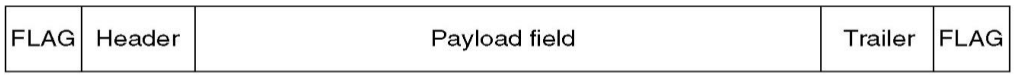
A character stream (a) Without errors (b) With one error

Problems

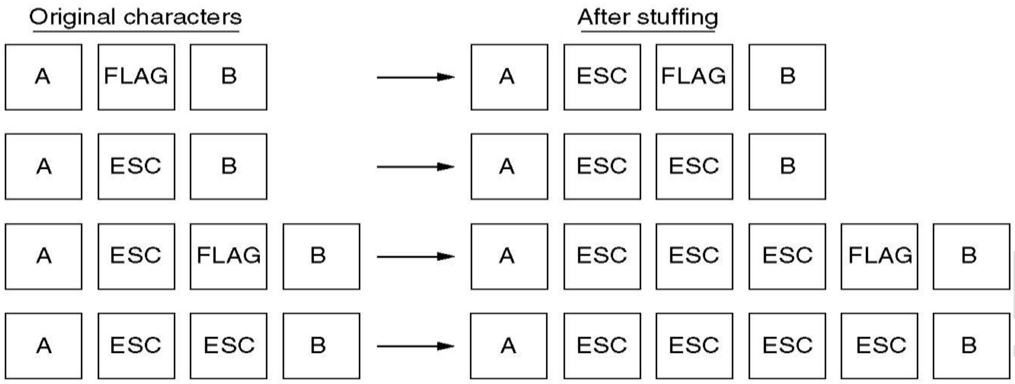
- If the count is damaged it is difficult to identify the start and end of frame.
- Even if with checksum, the receiver knows that the frame is bad there is no way to tell where the next frame starts. Asking for retransmission not possible because the start of the retransmitted frame is not known
- No longer used

Framing with byte/character

Each frame has special bytes called a flag byte, as both the starting and ending delimiter. In this way, if the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame. Two consecutive flag bytes indicate the end of one frame and start of the next one.



(a)



(b)

A serious problem occurs with this method is that the flag byte's bit pattern may occurs in the data. This situation will usually interfere with the framing.

Solution: Sender's data link layer insert a special escape byte (ESC) just before each "accidental" flag byte in the data. Receiver's data link layer removes the escape byte before the data are given to the network layer. This technique is called **byte stuffing or character stuffing**.

Problems: fixed character size, assumes character size to be 8 bits and can't handle heterogeneous environment.

Framing with bit stuffing

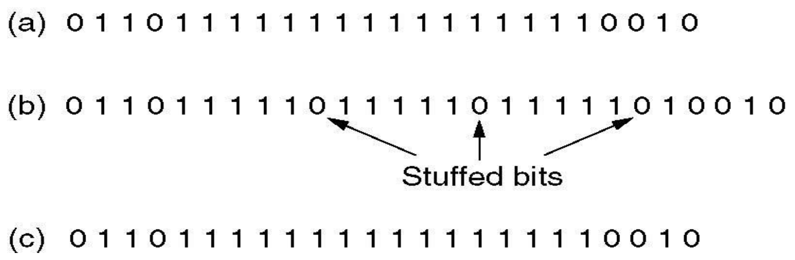
Each frame begins and ends with a special bit pattern, 01111110.

Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically deletes the 0 bit.

The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character.

Example


(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
Stuffed bits
(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 receiver's memory after de-stuffing.

With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data.

ERROR CONTROL

Positive and Negative Acknowledgement

To make sure all frames are eventually delivered to the network layer at the destination and in the proper order the receiver send back a special control frames bearing positive or negative acknowledgements about the incoming frames.

If the sender receives a positive acknowledgement about a frame, a frame has arrived safely. A negative acknowledgement means that something has gone wrong, and the **frame must be transmitted again.**

If there is hardware trouble, positive or negative acknowledgement will not reach the sender and it has to hang forever. In this case, timers are used.

Timers: When the sender transmits a frame, it also starts a timer. The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender.

If the frame will be correctly received, the acknowledgement will get back before the timer runs out, in which case the timer will be cancelled.

If either the frame or the acknowledgement is lost, the timer will go off, alerting the sender about problem and the sender just transmit the frame again.

Duplicates and sequence numbers: When frames may be transmitted multiple times the receiver will accept the same frame two or more times and pass it to the network layer more than once. To prevent this, each outgoing frame is assigned with sequence numbers so that the receiver can distinguish retransmissions from originals.

FLOW CONTROL

Making communication between speed mismatched networks is called as flow control. Two types of flow control are

- Feed back-based flow control
- Rate-based flow control

Feed back-based flow control

- The receiver sends back information to the sender about its current situation of handling data, and giving the sender permission to send more data or reduce data flow.

Rate-based flow control

- The protocol has a built in mechanism that limits the rate at which sender may transmit data without feedback from receiver.

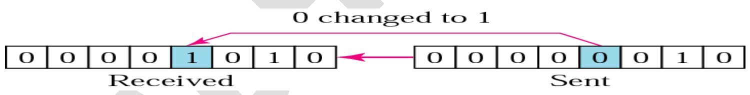
ERROR DETECTION AND CORRECTION

Data can be corrupted during its transmission. For reliable communication, errors must be detected and corrected. Error detection and correction are implemented either at the data link layer or the transport layer of the OSI model

Types of Error

1. Single bit error
2. Burst error

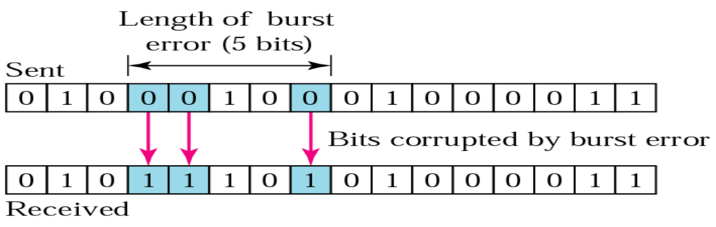
In a **single-bit error**, only one bit in the data unit has changed.



Single-bit error happens if data is sent using parallel transmission.

Burst Error

A burst error means that 2 or more bits in the data unit have changed.



Burst error doesn't mean that the errors occur in consecutive bits. Length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.

Burst error happen in a serial transmission. No. of bits affected depends on the data rate and duration of noise.

Example

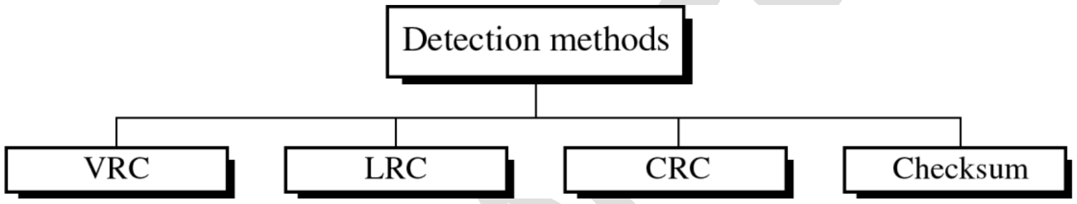
Data sent at 1 kbps, a noise of 1/100 seconds can affect (1000/100) 10 bits

Data sent at 1 mbps the same noise can affect 10,000 bits (1000000/100).

Error detection

Error detection uses the concept of <u>redundancy</u>, which means adding extra bits for detecting errors at the destination.

<u>Four types of Redundancy checks</u>



1. Vertical Redundancy Check
2. Longitudinal Redundancy Check
3. Cyclic Redundancy Check (CRC)
4. Checksum

Vertical Redundancy Check (VRC)

VRC is also known as Parity Check. In parity check, a parity bit is added to every data unit so that the total number of 1s is even (or odd for odd-parity).

Suppose the sender wants to send the word world. In ASCII the five characters are coded as

1110111 1101111 1110010 1101100 1100100

The following shows the actual bits sent (with redundant bit)

1110111<u>0</u> 1101111<u>0</u> 1110010<u>0</u> 1101100<u>0</u> 1100100<u>1</u>

When it is received, the receiver counts the 1s in each character. If no.of 1's in each character is even (6, 6, 4, 4, and 4) data are accepted.

Now suppose the word is corrupted during transmission.

111**1**1110   11011110   1110**1**100   11011000   11001001

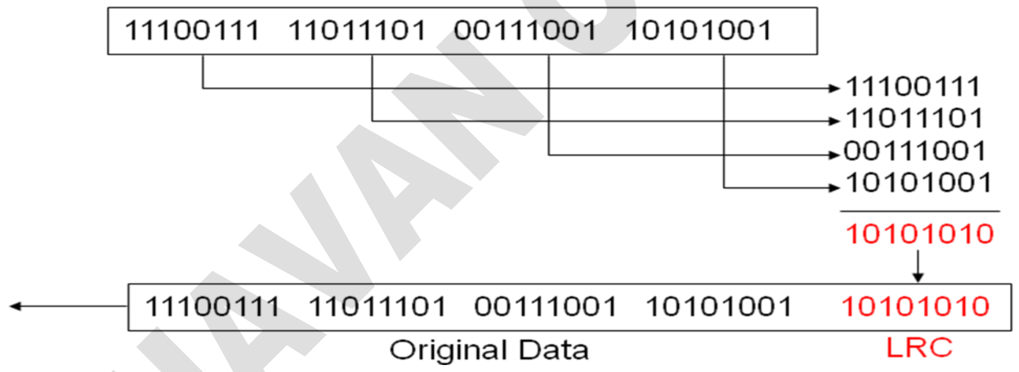
The receiver counts the 1s in each character and comes up with even and odd numbers (7, 6, 5, 4, and 4). The receiver knows that the data are corrupted, discards them, and asks for retransmission.

Simple parity check can detect all single-bit errors. It can detect burst errors only if the total number of errors in each data unit is odd.

Longitudinal Redundancy Check (LRC)

- Two-dimensional parity check

- Block of bits is divided into rows and a parity for each column is created and redundant row of bits is added to the whole block.



Performance of LRC

- Detects all burst errors up to length n (number of columns)
- If two bits in one data unit are changed and two bits in exactly same positions in another data unit are also damaged, LRC checker will not detect errors.

Example

Suppose the following block is sent:

10101001   00111001   11011101   11100111   10101010

However, it is hit by a burst noise of length 8, and some bits are corrupted.

10100011   10001001   11011101   11100111   10101010

When the receiver checks the parity bits, some of the bits do not follow the even-parity rule and the whole block is discarded.

10100011   10001001   11011101   11100111   10101010

Cyclic Redundancy Check (CRC)

1. Most powerful
2. Is based on binary division (divisor and remainder).
3. Instead of adding bits to achieve desired parity (as VRC, LRC), a sequence of redundant bits called CRC remainder is appended to the end of a data unit.

CRC remainder has two qualities. CRC remainder is generated using modulo-2 division

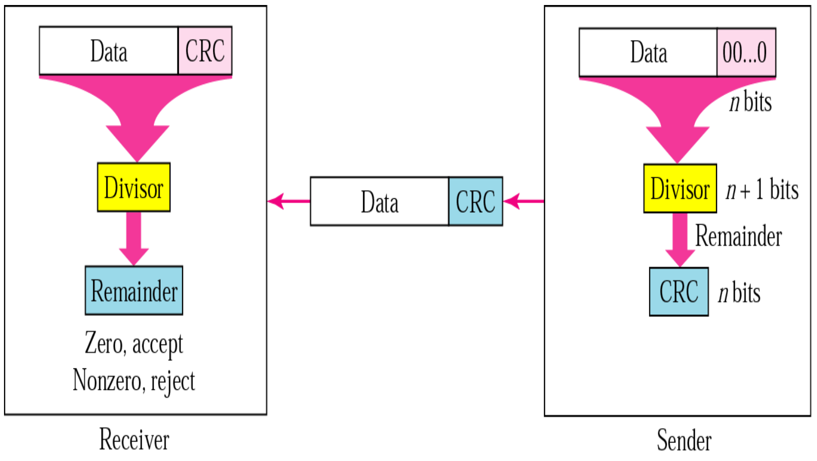
1. It must have exactly one less bit than the divisor.
2. Appending it to the end of data string must make it exactly divisible by divisor.

Basic steps

1. String of n 0's is appended to data unit.

   n- One less than the no. of bits in the predetermined divisor, which is n+1 bits
2. Newly enlarged data unit is divided by the divisor using binary division.
3. CRC remainder of n bits derived in step2 replaces at the end of data unit. CRC remainder may consist of all zeros.
4. The message (with the remainder) is transmitted to the receiver.
5. The receiver treats whole string as data unit and divides it by the same divisor that was used to find CRC remainder.

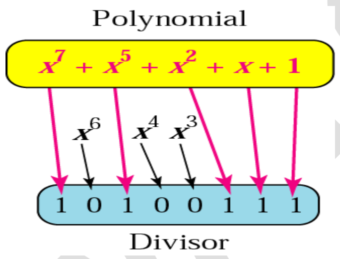
If the remainder is zero then there was no error during transmission so data is accepted by discarding CRC remainder. If the remainder is not equal to zero then there was an error during transmission. So data is rejected as error.

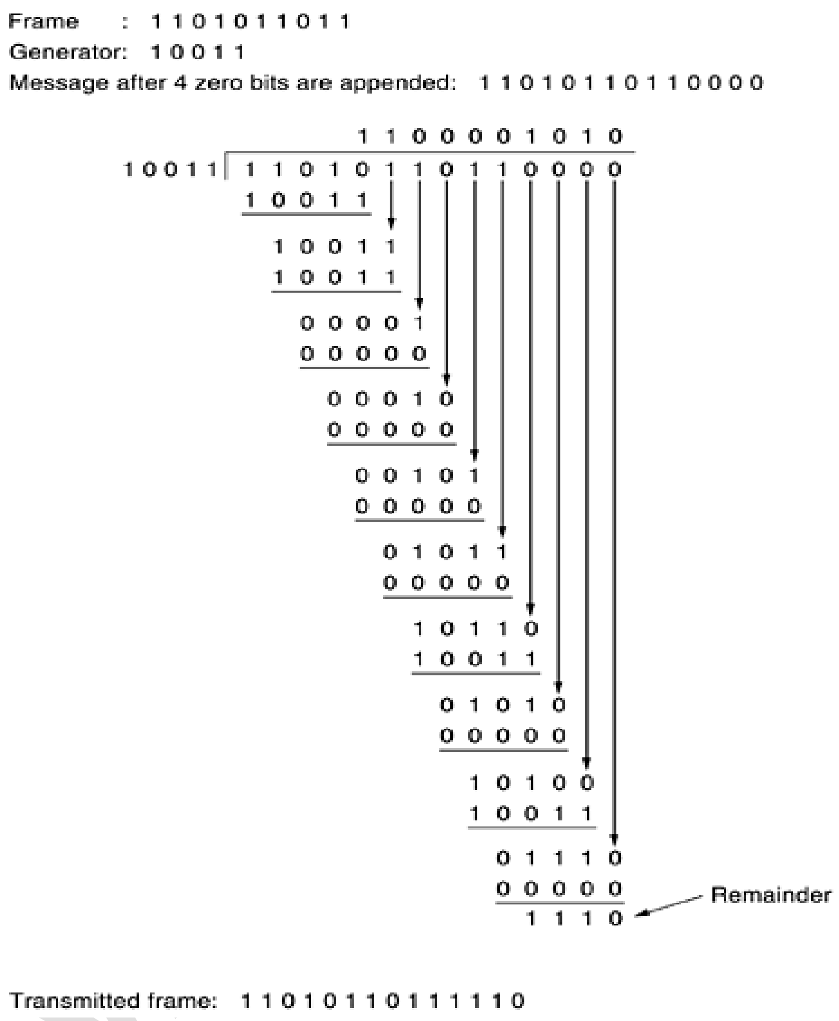


CRC Generator (the divisor) is created based on polynomial. A polynomial is selected with the following properties. It should not be divisible by x and it should be divisible by (x+1).

$$\boxed{x^7 + x^5 + x^2 + x + 1}$$

The relationship of a polynomial to its corresponding binary representation is





Data with CRC Remainder is divided at the receiver-side using same divisor created at sender-side. If data is not corrupted surely the remainder will be zero.

Performance of CRC

1. CRC can detect all burst errors that affect an odd number of bits.
2. CRC can detect all burst errors of length less than or equal to the degree of the polynomial.

Checksum

– Used by higher layers

– It detects all errors involving an odd number of bits, as well as even number of bits.

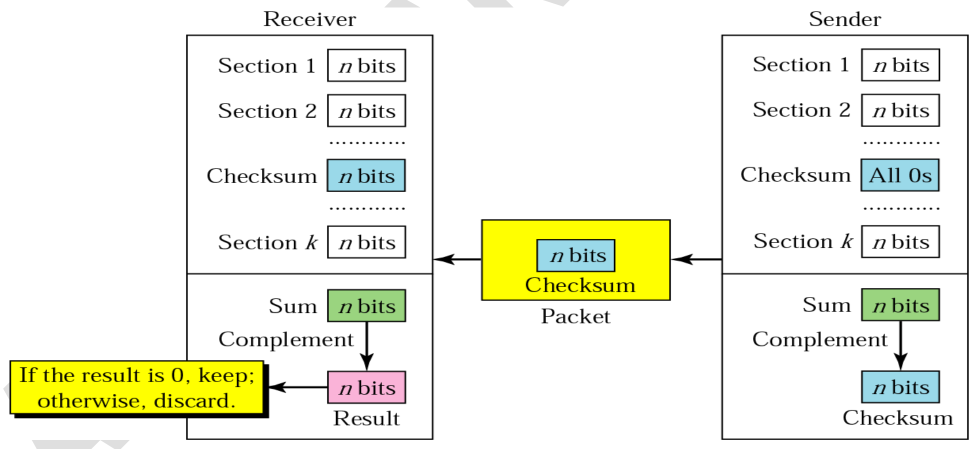
– It retains all carries during addition.

The sender follows these steps:

- The data unit is divided into k sections, each of n bits.
- All sections are added to get the sum.
- The sum is complemented and becomes the checksum.
- The checksum is sent with the data.

The receiver follows these steps:

- The data unit received is divided into k sections, each of n bits.
- All sections are added to get the sum.
- The sum is complemented.
- If the result is zero, the data are accepted: otherwise, rejected.



Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.

10101001 00111001

The numbers are added using one's complement

10101001

00111001

Sum 11100010

Checksum 00011101

The pattern sent is 10101001 00111001 00011101

Now suppose the receiver receives the pattern sent and there is no error.

10101001 00111001 00011101

When the receiver adds the three sections, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

10101001

00111001

00011101

Sum 11111111

Complement 00000000 means that the pattern is OK.

Now suppose there is a burst error of length 5 that affects 4 bits.

10101111 11111001 00011101

When the receiver adds the three sections, it gets

1 0 1 0 1 1 1 1

1 1 1 1 1 0 0 1

0 0 0 1 1 1 0 1

Partial Sum 1 1 1 0 0 0 1 0 1

Carry 1

Sum 1 1 0 0 0 1 1 0

Complement 00111001 the pattern is corrupted.

Error Correction can be handled in two ways.

1. When an error is discovered, the receiver can have the sender to retransmit the entire data unit.

2. A receiver can use an error-correcting code, which automatically corrects certain errors.

ELEMENTARY DATALINK PROTOCOLS

- An Unrestricted Simplex Protocol
- A Simplex Stop-and-Wait Protocol
- A Simplex Protocol for a Noisy Channel

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

Simplex stop and wait protocol

```
/* 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 a Noisy Channel

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

```
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 PROTOCOLS

- When there is a need for transmitting data in both directions, having two separate communications channels and use each one for one direction is not a good solution.
- A better idea is to use the same circuit for data in both directions.

Piggybacking

- The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**.

  Advantage: better use of the available channel bandwidth.
- When a data frame arrives, instead of immediately sending a separate control frame, the receiver waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame.

Waiting time

- Receiver waits for a fixed number of milliseconds, if a new packet arrives quickly, the acknowledgement is piggybacked onto it, and otherwise if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame.

Sender & Receiver window

In all sliding window protocols

- The sender maintains a set of sequence numbers corresponding to frames it is permitted to send. 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.

Sender’s window

- It contains frames sent but not yet acknowledged. New packets from the host cause the upper edge inside sender window to be incremented. Acknowledged frames from the receiver cause the lower edge inside window to be incremented.

  1) All frames in the sender’s window must be saved for possible retransmission and we need one timer per frame in the window.

  2) If the maximum sender window size is B, the sender needs B buffers.

  3) If the sender window gets full (i.e., reaches its maximum window size, the protocol must shut off the host (the network layer) until buffers become available.

Receiver window

Frames received with sequence numbers outside the *receiver window* are not accepted.

The receiver window size is normally static.

The set of acceptable sequence numbers is rotated as “acceptable” frames arrive.

1. **Go Back N**

Since receiver’s window size is 1 *the protocol only accepts frames in order*. This is referred to as Go Back N.

Disadvantage

It needlessly repeats frames those who are received after a corrupted frame.

2. selective repeat (also called selective reject)

Window size at most half the range of sequence numbers. Sender only retransmits messages which were lost

Advantage

Higher efficiency in noisy channels

Disadvantage

More buffers, more complex algorithm, costs more