Data Link Layer
Position of the data-link layer

**Network layer**

**Data link layer**

- Packetizing
- Flow control
- Media access control
- Addressing
- Error control

**Physical layer**

**LANs**

**WANs**
Data link layer duties

- Packetizing
- Addressing
- Error control
- Flow control
- Access control
LLC and MAC sublayers

- Logical link control (LLC)
- Media access control (MAC)
- Physical layer

Transmission medium

IEEE standard

Internet model
IEEE standards for LANs

802.2 Logical link control (LLC)

802.3 CSMA/CD
802.4 Token bus
802.5 Token ring
802.6 DQDB

... 802.11 Wireless ...

Project 802
DATA LINK LAYER DESIGN

ISSUES

1. Provide service interface to the network layer
2. Dealing with transmission errors
3. Regulating data flow
   • Slow receivers not swamped by fast senders
Functions of the Data Link Layer (2)

Relationship between packets and frames.
Services Provided to Network Layer

a) Unacknowledged connectionless service:
   - No logical connection is established before hand 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.
   - Ethernet (coax or optical fiber cable) is a good example:
     - Appropriate when the error rate is very low, so recovery is left to higher layers.
     - Appropriate for real-time traffic, such as voice, in which late data are worse than bad data

b) Acknowledged connectionless service
   - No logical connections used, but each frame sent is individually acknowledged.
   - 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.
   - Eg: -802.11 (WiFi)

c) Acknowledged connection oriented service
Services Provided to Network Layer

(a) Virtual communication.
(b) Actual communication.
Placement of the data link protocol.
Framing

a) When the stream of bits is received on the destination point, it is not guaranteed to be error free.

b) The number of bits received might be less, equal or more than the number of bits transmitted and may have different values.

c) It is up to data link layer to detect and, if necessary, correct errors.

d) The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame and any changes in the value of checksum (it is calculated again at the receiver’s end) can assure the changes in the frame.
Framing techniques

a) Character Count
b) Flag bytes with byte stuffing
c) Starting and ending flags, with bit stuffing
Framing

A character stream. (a) Without errors. (b) With one error.
(a) A frame delimited by flag bytes.
(b) Four examples of byte sequences before and after stuffing.
Bit stuffing
(a) The original data.
(b) The data as they appear on the line.
(c) The data as they are stored in receiver’s memory after destuffing.
Error Control

a) Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination and in the proper order.

b) The usual way to ensure reliable delivery is to provide the sender with some feedback in the form of acknowledgement about what is happening at the other end of the line.

c) Use of timers in the case when the packet is totally lost during its transmission and the receiver has no idea about the packet and hence no acknowledgment.

d) Use of sequence numbers to deal with the problem in case multiple frames have arrived at the receiver’s end.
Flow control

a) Difference in the speed of the two communicating parties can cause the loss of packets (if the sender keeps on sending the packets and the receiver is unable to process them as quickly).

b) Two approaches to manage this: Feedback based flow control and Rate based flow control.

c) For example, when a connection is set up, the receiver might say: "You may send me n frames now, but after they have been sent, do not send any more until I have told you to continue."
Error Detection and Correction
Note:

Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.
10.1 Types of Error

Single-Bit Error

Burst Error
In a single-bit error, only one bit in the data unit has changed.
10.1 Single-bit error

0 changed to 1

0 0 0 0 1 0 1 0

Received

0 0 0 0 0 0 1 0

Sent
A burst error means that 2 or more bits in the data unit have changed.
10.2 Burst error of length 5

Length of burst error (5 bits)

Sent

0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1

Received

0 1 0 1 1 1 0 1 0 1 0 0 0 0 0 1 1

Bits corrupted by burst error
10.2 Detection

Redundancy

• Parity Check

• Cyclic Redundancy Check (CRC)

• Checksum
Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.
10.3 Redundancy

Receiver node

Data: 10100000000101010

Reject data No

OK?

Yes

Data & redundancy: 10100000000101010

Sender node

Data: 10100000000101010

Data & redundancy: 10100000000101010

Medium
Figure 10.4 XORing of two single bits or two words

\[
\begin{array}{c|c|c|c}
0 & 0 \oplus 0 = 0 & 1 \oplus 1 = 0 \\
0 & 0 \oplus 1 = 1 & 1 \oplus 0 = 1 \\
\end{array}
\]

a. Two bits are the same, the result is 0.

\[
\begin{array}{c|c|c|c|c|c}
1 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 \\
\hline
0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

b. Two bits are different, the result is 1.

c. Result of XORing two patterns

**Note**

In modulo-N arithmetic, we use only the integers in the range 0 to N –1, inclusive.
In block coding, we divide our message into blocks, each of \( k \) bits, called **datawords**. We add \( r \) redundant bits to each block to make the length \( n = k + r \). The resulting \( n \)-bit blocks are called **codewords**.

**Topics discussed in this section:**
Hamming Distance
Minimum Hamming Distance
Figure 10.5 Datawords and codewords in block coding

$2^k$ Datawords, each of $k$ bits

$2^n$ Codewords, each of $n$ bits (only $2^k$ of them are valid)
<table>
<thead>
<tr>
<th>Datawords</th>
<th>Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>01</td>
<td>011</td>
</tr>
<tr>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>11</td>
<td>110</td>
</tr>
</tbody>
</table>
Figure 10.7  Structure of encoder and decoder in error correction
The Hamming distance between two words is the number of differences between corresponding bits.
Let us find the Hamming distance between two pairs of words.

1. The Hamming distance $d(000, 011)$ is 2 because

   $000 \oplus 011$ is 011 (two 1s)

2. The Hamming distance $d(10101, 11110)$ is 3 because

   $10101 \oplus 11110$ is 01011 (three 1s)
The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.
To guarantee correction of up to $t$ errors in all cases, the minimum Hamming distance in a block code must be $d_{\text{min}} = 2t + 1$. 

**Note**
10.4 Detection methods

- Parity check
- Cyclic redundancy check
- Checksum
10.5 **Even-parity concept**

Receiver node:
- Drop parity bit and accept data
- Reject data
- **Even?**
  - Yes
  - No
- Count bits
- Bits

Sender node:
- **Calculate parity bit**
- Data: 1100001
- 1

Transmission Medium
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).
Example 1

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

11101110 11011110 11100100 11011000 11001001
**Example 2**

Now suppose the word *world* in Example 1 is received by the receiver without being corrupted in transmission.

```
11101110 11011110 11100100 11011000 11001001
```

The receiver counts the 1s in each character and comes up with even numbers (6, 6, 4, 4, 4). The data are accepted.
Example 3

Now suppose the word world in Example 1 is corrupted during transmission.

11111110 11011110 11101100 11011000 11001001

The receiver counts the 1s in each character and comes up with even and odd numbers (7, 6, 5, 4, 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.

If the error is even in that case it is not able to detect error in burst error case.
In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.
10.6 Two-dimensional parity

Two-dimensional parity

Original data

\[
\begin{array}{cccc}
1100111 & 1011101 & 0111001 & 0101001 \\
\end{array}
\]

Row parities

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
\end{array}
\]

Column parities

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
\end{array}
\]

Data and parity bits

\[
\begin{array}{cccc}
11001111 & 10111011 & 01110010 & 01010011 \\
\end{array}
\]
b. One error affects two parities

c. Two errors affect two parities

d. Three errors affect four parities

e. Four errors cannot be detected
Example 4

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
10.7 Cyclic Redundancy Check (CRC) generator and checker

**Detection**

The diagram illustrates the process of generating and checking cyclic redundancy check (CRC) codes. The sender takes the data, divides it by a predetermined divisor, and appends the remainder to the data to form the transmitted message. Upon reception, the receiver divides the received data by the same divisor and compares the remainder with the transmitted CRC. A zero remainder indicates that the data was transmitted without errors, while a nonzero remainder indicates an error in transmission.

**Sender**
- **Data**: Input data
- **00...0**: Additional bit(s) for CRC generation
- **CRC**: Cyclic redundancy check code
- **n + 1 bits**: Total bits including CRC
- **Divisor**: Predefined divisor for the cyclic redundancy check
- **Remainder**: Result of the division

**Receiver**
- **Data**: Received data
- **CRC**: Cyclic redundancy check code
- **Zero, accept**: If the remainder is zero, the data is accepted as error-free
- **Nonzero, reject**: If the remainder is nonzero, the data is rejected as errored
Detection  
Cyclic Redundancy Check

Binary division in a CRC generator

# A string of $n$ 0’s is appended to the data unit where $n$ is 1 less than the no. of bits in divisor.

# Newly elongated data unit is divided by divisor, remainder is the CRC.

# CRC of $n$ bits replaces the appended 0s at the end of the data unit.

Modulo-2Arithmetic
Detection

Cyclic Redundancy Check

Binary division in CRC checker

# After receiving the data appended with the CRC, it does the same modulo-2 division.

# If the remainder is all 0’s, the CRC is dropped and the data are accepted.

# Otherwise data is discarded.

---

When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
A polynomial

\[ x^7 + x^5 + x^2 + x + 1 \]
10.11 A polynomial representing a divisor

Polynomial: \( x^7 + x^5 + x^2 + x + 1 \)

Divisor: 
```
1 0 1 0 0 1 1 1
```
### Table 10.1 Standard polynomials

<table>
<thead>
<tr>
<th>Name</th>
<th>Polynomial</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC-8</td>
<td>$x^8 + x^2 + x + 1$</td>
<td>ATM header</td>
</tr>
<tr>
<td>CRC-10</td>
<td>$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$</td>
<td>ATM AAL</td>
</tr>
<tr>
<td>ITU-16</td>
<td>$x^{16} + x^{12} + x^5 + 1$</td>
<td>HDLC</td>
</tr>
</tbody>
</table>
| ITU-32  | $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10}$  
  $+ x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ | LANs        |
Example 5

It is obvious that we cannot choose $x$ (binary 10) or $x^2 + x$ (binary 110) as the polynomial because both are divisible by $x$. However, we can choose $x + 1$ (binary 11) because it is not divisible by $x$, but is divisible by $x + 1$. We can also choose $x^2 + 1$ (binary 101) because it is divisible by $x + 1$ (binary division).
Example 6

The CRC-12

\[ x^{12} + x^{11} + x^3 + x + 1 \]

which has a degree of 12, will detect all burst errors affecting an odd number of bits, will detect all burst errors with a length less than or equal to 12, and will detect, 99.97 percent of the time, burst errors with a length of 12 or more.
The sender follows these steps:

• The unit is divided into $k$ sections, each of $n$ bits.

• All sections are added using one’s complement 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 unit is divided into $k$ sections, each of $n$ bits.

• All sections are added using one’s complement to get the sum.

• The sum is complemented.

• If the result is zero, the data are accepted; otherwise, rejected.
10.12 Checksum

**Receiver**

<table>
<thead>
<tr>
<th>Section 1</th>
<th>$n$ bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 2</td>
<td>$n$ bits</td>
</tr>
<tr>
<td>Checksum</td>
<td>$n$ bits</td>
</tr>
<tr>
<td>Section $k$</td>
<td>$n$ bits</td>
</tr>
</tbody>
</table>

**Sender**

<table>
<thead>
<tr>
<th>Section 1</th>
<th>$n$ bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 2</td>
<td>$n$ bits</td>
</tr>
<tr>
<td>Checksum</td>
<td>All 0s</td>
</tr>
<tr>
<td>Section $k$</td>
<td>$n$ bits</td>
</tr>
</tbody>
</table>

- **Checksum**
- **Packet**

**Description**

- **Sum** $n$ bits
- **Complement** $n$ bits
- **Result** $n$ bits

**Instructions**

- If the result is 0, keep; otherwise, discard.
10.13 Data unit and checksum

Sum Complement

$T$ $-T$

$-0$ $0$

Receiver $T$ $-T$

Sender
Example 7

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
Example 8

Now suppose the receiver receives the pattern sent in Example 7 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.

\[
\begin{array}{c c c c}
10101001 \\
00111001 \\
00011101 \\
\hline
\text{Sum} & 11111111 \\
\text{Complement} & 00000000
\end{array}
\]

\[\text{Complement 00000000 means that the pattern is OK.}\]
Example 9

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

\[ 10101\overline{11111001} \ 00011101 \]

When the receiver adds the three sections, it gets

\[ 10101111 \]
\[ 11111001 \]
\[ 00011101 \]

Partial Sum: \[ 1 \ 11000101 \]
Carry: \[ 1 \]
Sum: \[ 11000110 \]
Complement: \[ 00111001 \] the pattern is corrupted.
How can we represent the number 21 in one’s complement arithmetic using only four bits?

How can we represent the number \(-6\) in one’s complement arithmetic using only four bits?

**Sender site**

<table>
<thead>
<tr>
<th>Sum</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrapped sum</td>
<td>6</td>
</tr>
<tr>
<td>Checksum</td>
<td>9</td>
</tr>
</tbody>
</table>

**Receiver site**

<table>
<thead>
<tr>
<th>Sum</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrapped sum</td>
<td>15</td>
</tr>
<tr>
<td>Checksum</td>
<td>0</td>
</tr>
</tbody>
</table>

Details of wrapping and complementing:

\[
\begin{align*}
1 & 0 0 1 0 0 \\
\downarrow & \quad 1 0 \\
0 & 1 1 0 \\
1 & 0 0 1 \\
\end{align*}
\]

Details of wrapping and complementing:

\[
\begin{align*}
1 & 0 1 1 0 1 \\
\downarrow & \quad 1 0 \\
1 & 1 1 1 1 \\
0 & 0 0 0 \\
\end{align*}
\]
10.3 Correction

The correction of errors is more difficult than the detection.

In error correction, we need to know the exact number of bits that are corrupted and more importantly, their location in the message.

- Retransmission
- Forward Error Correction
- Burst Error Correction
Single-bit error correction

To correct an error, the receiver reverses the value of the altered bit. To do so, it must know which bit is in error.

Number of redundancy bits needed

- Let data bits = \( m \)
- Redundancy bits = \( r \)

\[ \therefore \text{Total message sent} = m + r \]

The value of \( r \) must satisfy the following relation:

\[ 2^r \geq m + r + 1 \]
Error Correction

Data \((m)\) bits

\[
\begin{array}{c}
\quad
dots
\quad
\end{array}
\]

Redundancy \((r)\) bits

Total \(m + r\) bits
\[2^r \geq m + r + 1\]

**Table 10.2 Data and redundancy bits**

<table>
<thead>
<tr>
<th>Number of data bits (m)</th>
<th>Number of redundancy bits (r)</th>
<th>Total bits (m + r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>
10.14 Positions of redundancy bits in Hamming code

```
<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>d</td>
<td>d</td>
<td>r_8</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>r_4</td>
<td>d</td>
<td>r_2</td>
<td>r_1</td>
</tr>
</tbody>
</table>
```

Redundancy bits
Hamming Code

$r_1$ will take care of these bits

$11$ $1001$ $0111$ $0101$ $0011$ $0001$

$d$ $d$ $d$ $r_8$ $d$ $d$ $d$ $r_4$ $d$ $r_2$ $r_1$

$r_2$ will take care of these bits

$10111010$ $01110110$ $00110010$

$d$ $d$ $d$ $r_8$ $d$ $d$ $d$ $d$ $r_4$ $d$ $r_2$ $r_1$
Hamming Code

$r_4$ will take care of these bits

011101100101 0100
7 6 5 4

d d d d $r_8$ d d d d $r_4$ d $r_2$ $r_1$

$r_8$ will take care of these bits

101110101001 1000
11 10 9 8

d d d d $r_8$ d d d d $r_4$ d $r_2$ $r_1
10.15 *Redundancy bits calculation*

$r_1$ will take care of these bits.

\[
\begin{array}{ccccccc}
11 & 9 & 7 & 5 & 3 & 1 \\
\text{d} & \text{d} & \text{d} & \text{r}_8 & \text{d} & \text{d} & \text{r}_4 & \text{d} & \text{r}_2 & \text{r}_1
\end{array}
\]

$r_2$ will take care of these bits.

\[
\begin{array}{cccccccc}
11 & 10 & 7 & 6 & 3 & 2 \\
\text{d} & \text{d} & \text{d} & \text{r}_8 & \text{d} & \text{d} & \text{d} & \text{d} & \text{r}_4 & \text{d} & \text{r}_2 & \text{r}_1
\end{array}
\]

$r_4$ will take care of these bits.

\[
\begin{array}{cccccccc}
7 & 6 & 5 & 4 \\
\text{d} & \text{d} & \text{d} & \text{r}_8 & \text{d} & \text{d} & \text{d} & \text{d} & \text{r}_4 & \text{d} & \text{r}_2 & \text{r}_1
\end{array}
\]

$r_8$ will take care of these bits.

\[
\begin{array}{ccccccc}
11 & 10 & 9 & 8 \\
\text{d} & \text{d} & \text{d} & \text{r}_8 & \text{d} & \text{d} & \text{d} & \text{r}_4 & \text{d} & \text{r}_2 & \text{r}_1
\end{array}
\]
Example of redundancy bit calculation

Data: 1001101

Code: 10011100101
10.17 Error detection using Hamming code

The bit in position 7 is in error.
10.18  Burst error correction example

Error → 1111?000011
Error → 1010?011111
Error → 011?1011001
Error → 011?1010110
Error → 011?1001111

Received data

Direction of transmission

Data in transition

1 1 1 1 1 0 0 0 0 1 1
1 0 1 0 1 0 1 1 1 1 1
1 1 1 1 1 0 0 1 1 0 0
0 1 1 0 1 0 1 1 0 0 1
0 1 1 0 1 0 1 0 1 1 0
0 1 1 1 1 0 0 1 1 1 1

Data before being sent

11111000011
10101011111
11111001100
01101011001
01101010110
01111001111