Responsibilities of the Physical and Data Link Layers

The Physical and Data Link layers have strongly related responsibilities. Unlike other adjacent layers, it's not possible to simply replace an implementation of one layer with another.

**Physical Layer responsibilities:**
- Provides the physical topology of the network
- Provides the transfer medium (such as copper or optical cable)
- Encodes data into a transmission signal supported by the medium
- Generates or amplifies the signal along the medium
- Monitors for (detects) transmission errors
- Detects arriving signal levels to synchronize speed and timing

**Data Link Layer responsibilities:**
- Constructs data frames using the format of the Physical Layer
- Creates Cyclic Redundancy Codes (CRC) information
- Checks for arriving errors using CRC information
- Initiates and arbitrates the link to reduce interruption and contention
- Examines arriving and passing device addresses in data
- Retransmits data if there is an error
- Acknowledges receipt of a frame
Metrics Of Network Performance

There are two components to performance - latency and bandwidth. We always want high bandwidth and low latency, but can't always get both together.

- **Latency or propagation delay** is the amount of time it takes for the first bit to reach its destination. *Round-trip time* (RTT) is twice the latency - the time for one bit to travel to the destination, and the first bit in reply to return.
- **Bandwidth** is the number of bits that can be fit through a network connection, per unit time. Historically, i.e. without any data compression, bandwidth has been related to frequency within the medium.
- **Throughput** is usually the measured number of bits per second (achieved), while bandwidth is the nominal (peak) number of bits per second possible. High bandwidth does not imply low latency: you can broadcast many Mbps over a satellite connection, but it will take many milliseconds for the first data to arrive.

A note about prefixes: In networking, the prefix Mega means $10^6$, at least when referring to throughput and bandwidth. We always use powers of 10 when referring to Hertz. When talking about the size of a message, buffer, file, or other computer-storage item, we use Mega to mean $2^{20}$. Similar rules apply for the prefixes Kilo and Giga.

of: [Binary prefix](https://en.wikipedia.org/wiki/Binary_prefix)

<table>
<thead>
<tr>
<th>Example: a car can carry 4 people to Bunbury in 2 hours. Thus the bandwidth is 2 people/hour.</th>
<th>A bus can take 60 passengers and arrive in 2 hours. Thus the bandwidth is 30 people/hour, but the latency is still 2 hours.</th>
</tr>
</thead>
</table>

The **latency of a message** is the total time for the whole message to arrive:

$$
T_{\text{Latency}} = T_{\text{Propagation}} + T_{\text{Transmit}} + T_{\text{Queue}}
$$

$$
T_{\text{Propagation}} = \frac{\text{distance}}{\text{speed-of-propagation-in-medium}}
$$

$$
T_{\text{Transmit}} = \frac{\text{message-size}}{\text{bandwidth}}
$$

$$
T_{\text{Queue}} = \text{time-spent-in-local-and-remote-operating-system-and-switch-queues}
$$
The Physical Layer and Transmission Errors

<table>
<thead>
<tr>
<th>Speed</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telephone cable: $10^5 - 10^8$bps</td>
<td>1 in $10^5$ bits</td>
</tr>
<tr>
<td>Shielded copper cable: $10^7 - 10^9$bps</td>
<td>1 in $10^{12} - 10^{13}$ bits</td>
</tr>
<tr>
<td>Optical fibre: $10^9 - 10^{14}$bps</td>
<td>1 in $10^{16} - 10^{17}$ bits</td>
</tr>
<tr>
<td>Wireless systems: $10^6 - 10^{12}$bps</td>
<td>1 in $10^5 - 10^9$ bits</td>
</tr>
</tbody>
</table>

Rates of transmission errors are described by their probability of occurring, or by their expected bit-error-rate (BER).

Transmission errors usually occur in bursts and are caused by:

- Thermal background noise
- Impulse noise (~10ms duration)
- Distorted frequency
- Crosstalk
- Reduction of dynamic range and noise (companding)

Over a particular time interval, a burst begins at the first bit to be corrupted (inverted), and ends at the last bit corrupted - the bits between are not necessarily all modified.
How Data is Placed in Frames

Simple timing gaps between frames cannot be used as all hardware devices (as part of the physical layer) run at slightly different speeds; resulting in skewing if time is relied upon.

A first attempt at a solution involves **counting** the size of a frame and placing this count in the data link header - blah!

This naive solution highlights problems with synchronization - assuming the receiver can detect a problem, how much of the arriving bit stream should be skipped to find the next frame?
How Data is Placed in Frames, continued

To overcome synchronization problems, special character sequences are often used to prefix and suffix (envelope) the data. As these special characters may themselves be required data we must "escape" their special meaning (particularly true for "binary" data such as floating point numbers, executables, JPEGs and MP3s).

This process is termed **character stuffing** (DLE=020h, STX=002h, ETX=003h). (On a Linux or OS-X machine, `executeman ascii`)

Consider the following example, transmitting 3 characters: [A, DLE, B]

![Diagram](image)

A lower-level approach, **bit-stuffing**, overcomes the reliance on using the ASCII codes. Each frame is now enveloped in pairs of "flag" patterns 01111110.

![Diagram](image)

If this flag pattern appears in the data, the stuffed sequence 01111010 is transmitted.
Phase Encoding of Signals

To be able to detect collisions on LANs, we need to understand how digital signals are encoded on the physical medium.

A digital signal is a sequence of discrete, discontinuous pulses, or signal elements.

If all signal elements have the same (voltage) sign, they are termed unipolar.

The modulation rate of a LAN is the number of signal elements (transitions) per second, or the baud rate.

To correctly interpret a signal, the receiver must know the length (time) of each signal element and the expected voltage levels of the bits 0 and 1.

The two simplest encoding schemes are the:

- Non-return to Zero Level (NRZ-L) and
- Non-return to Zero Inverted (NRZ-I).

NRZ-I is an example of differential encoding in which the signal transition, rather than the signal level, indicates the value of each bit.
Phase Encoding of Signals, continued

NRZ-L and NRZ-I are simple schemes, but unfortunately offer no possible synchronization.

Another scheme, biphase encoding, ensures that there will be a transition in the middle of each bit.

With traditional Manchester encoding the mid-bit transition provides a clocking mechanism (as well as the data).

With Differential Manchester the mid-bit transition provides the clocking and the initial transition provides the data.

Biphase encoding schemes have (at least) one transition per bit time, thus the maximum modulation rate, or baud rate, is twice that for NRZ-L and NRZ-I.
Error Detection and Correction

Data may be modified so that errors can either be:

- detected (i.e. you can only tell that the data is wrong), or
- corrected (i.e. you can unambiguously tell what the data should have been, and hence you can confidently correct it).

Correction is required where communication must be simplex (only possible in one direction), but correction is expensive. A good example of its need is between Earth and inter-planetary spacecraft. Error correction by the receiver is referred to as forward error correction, whereas re-transmission schemes are referred to as reverse error correction.

Codewords are constructed consisting of both data and check bits.

The Hamming distance between two codewords consists of the number of bit positions in which they differ. The difference is performed Modulo-2 or EXCLUSIVE-OR:

e.g. 1 ⊕ 1 = 0 ⊕ 0 = 0
1 ⊕ 0 = 0 ⊕ 1 = 1

\[
\begin{array}{c}
10011101 \\
10111110 \\
\hline
00100011
\end{array}
\]

Here, the Hamming distance is 3.

The Hamming distance of a code is the minimum Hamming distance between any two words in that code.
Error Detection and Correction, continued

To detect: $\delta$ errors, a distance of $\delta + 1$ is required.

\[\text{e.g. 1 error requires no word with a distance of 1 from a valid word.}\]

To correct: $2\delta + 1$ errors are required so that even with $\delta$ errors, the damaged codeword is the closest to one valid codeword.

Some examples:

Consider a simple encoding of ASCII characters, where we use a parity code [see editions of Tanenbaum \(\geq 5th\)].

We have 7 data bits and a single parity-code bit. Now, what happens if any single bit is corrupted?

- What is the distance of the parity-code?
- What type of errors can it detect?
- What type of errors can it correct?

Consider a code whose only valid code words are:

\[
\begin{align*}
00000000 & \\ 111110000 & \\ 000011111 & \\ 111111111
\end{align*}
\]

- What is the distance of this code?
- What type of errors can it detect?
- What type of errors can it correct?
Hamming’s Correction of Single-Bit Errors

Let’s say that a single transmission consists of \( m \) bits for the message, and \( r \) bits of seemingly redundant information. We thus transmit \( n = m + r \) bits when transmitting a message.

The critical question is "how much additional information (the redundant bits) do we need to transmit so that the receiver can correct errors?"

Each of the \( 2^m \) possible message words has \( n \) illegal code words which are a distance 1 from it. Therefore each message word requires \( n + 1 \) distinct bit patterns (1 legal one, and \( n \) illegal ones).

In 1950, mathematician and Turing Award winner Richard Hamming developed a method which achieves the lower bound of:

\[ m + 1 \leq 2^r. \]

Given a code word of 7 data bits and 4 check bits, we number the code word from 1 from the left hand side.

Each bit whose ordinal position is a power of 2 \( \{1,2,4,8,...\} \) is a check bit and forces the parity of some "collection" of bits including itself. Parity may be forced to be either even or odd.
Hamming’s Correction of Single Errors, continued

A data bit contributes to the parity of all the bits in its decomposition into a sum of powers of 2.

For example:

\[
\begin{align*}
3 &= 2 + 1 \\
7 &= 4 + 2 + 1 \\
11 &= 8 + 2 + 1
\end{align*}
\]

Making a list of these:

\[
\begin{align*}
1 & \text{ is contributed to by } 3, 5, 7, 9, 11 \\
2 & \text{ is contributed to by } 3, 6, 7, 10, 11 \\
4 & \text{ is contributed to by } 5, 6, 7 \\
8 & \text{ is contributed to by } 9, 10, 11
\end{align*}
\]

We then put the data bits in their positions and calculate each check bit.
Hamming's Correction of Single Errors, continued

For example, ASCII character 'c' = 1100011, using even parity:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

check 1  X    X    X    X    X
check 2  X    X    X    X    X
check 3  X    X    X    X    X
check 4  X    X    X    X    X

The resulting codeword is hence 1111000011.

We then transmit this codeword across the network.
Hamming's Correction of Single Errors, continued

When a codeword is received at the other end we must check (and possibly correct) it.

To correct when it arrives:

Initialize a counter, c, to 0.

Examine each check bit in position \( k = \{1, 2, 4, 8\} \) to see if it has the correct parity.

If not, add \( k \) to \( c \).

When all check bits have been examined, and counter \( c \) is 0, then codeword is correct. Otherwise, the incorrect bit is in position \( c \).

For example, if we transmit 11111000011 (from before), but we receive:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\( c = 0 \)

\( k = 1 \) parity for 3, 5, 7, 9, 11 wrong! so \( c + = k \) (=1)

\( k = 2 \) parity for 3, 6, 7, 10, 11 wrong! so \( c + = k \) (=3)

\( k = 4 \) parity for 5, 6, 7 correct

\( k = 8 \) parity for 9, 10, 11 wrong! so \( c + = k \) (=11)

Hence bit 11 is in error!
Cyclic Redundancy Codes (CRCs)

The extra bits used to detect errors are called checksum bits.

A checksum can be quite simple, for example "add-with-carry" all bytes in a message, or much more sophisticated as in the case of a cyclic redundancy code (or CRC).

Polynomial Codes

A polynomial is represented by a bit string with 1 for each power represented in the polynomial, and 0 otherwise.

e.g. \( x^4 + x^3 + 1 \) is represented as 11001

Polynomial arithmetic is performed Modulo-2 or EXCLUSIVE-OR

\[
\begin{align*}
1 \oplus 1 &= 0 \\
1 \oplus 0 &= 0 \\
0 \oplus 1 &= 1
\end{align*}
\]

Both the sender and receiver agree on a generator polynomial, \( G(x) \), with high and low order bits 1.

The message, \( M \), is also interpreted as a polynomial.

The checksum (when calculated) is appended to the message so that \( M + \text{checksum} \), known as the transmission \( T(x) \), is divisible by \( G(x) \).

On arrival we simply check that the received transmission is divisible by \( G(x) \).

[See A Painless Guide to CRC Error Detection Algorithms]

For examples of other types of checksum algorithms, see the Australian Securities & Investment Commission and the Australian Tax Office.
**Polynomial Codes, continued**

What types of errors can (and cannot) be detected with CRCs?

Let the error bits in $T_{\text{arrived}}(x)$ be $E(x)$. Each bit in $E(x)$ corresponds to a bit that has been inverted. If there are $k$ 1 bits in $E(x)$, then there have been $k$ single-bit errors.

Then, due to the properties of the Modulo-2 arithmetic:

$$\left( T(x) + E(x) \right) / G(x) = \left( T(x) / G(x) \right) + \left( E(x) / G(x) \right)$$

$$= 0 + \left( E(x) / G(x) \right)$$

We notice that if $G(x)$ divides $E(x)$ (because $G(x)$ is a factor of $E(x)$), then errors can go by undetected.

**Some important results:**

- If $G(x)$ has two or more terms all single bit errors can be detected.
- To detect double errors, $G(x)$ must not be divisible by $(x^k + 1)$ for $k$ up to some maximum frame length.
- To detect an odd number of errors, $G(x)$ should contain $(x+1)$ as a factor.
- Polynomials of degree $r$ will detect all burst errors of $\leq r$ bits.
Some Standard Polynomial Codes

16 bit checksums catch -

- all single and double bit errors
- all errors with an odd number of bits
- all burst errors <= 16 bits long
- 99.997% of burst errors = 17 bits
- 99.998% of burst errors >= 18 bits

\[ \text{CRC-12} = x^{12} + x^{11} + x^3 + x^2 + x + 1 \]

\[ \text{CRC-16} = x^{16} + x^{15} + x^2 + 1 \]

\[ \text{CRC-CCITT} = x^{16} + x^{12} + x^5 + 1 \]

\[ \text{CRC-32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \]

CRCs are often performed in hardware, for example the DEC-VAX architecture (mid-1980s) had an assembly language instruction to perform CRC-16 (!).

Today, all network interface cards (such as wired and wireless Ethernet) perform checksumming "on-board".