Modern Data Communications: Analog and Digital Signals, Compression, Data Integrity

Cristian S. Calude

July-August 2012

Goals

- Understand digital and analog signals
- Understand codes and encoding schemes
- Understand compression, its applications and limits
- Understand codes for error detection and correction

References

Factors determining data transmission

- cost of a connection
- amount of information transmitted per unit of time (bit rate)
- immunity to outside interference (noise)
- security (susceptibility to unauthorised “listening”, modification, interruption, or channel usage)
- logistics (organising the wiring, power, and other physical requirements of a data connection)
- mobility (moving the station)

Analog and digital signals

Connected devices have to “understand” each other to be able to communicate.

*Communication standards* assure that communicating devices represent and send information in a “compatible way”.

There are two types of ways to transmit data:

- **via digital signals**, which can be represented either electronically (by sequences of specified voltage levels) or optically,
- **via analog signals**, which are formed by continuously varying voltage levels.

Digital signals are graphically represented as a square wave: the horizontal axis represents time and the vertical axis represents the voltage level.
The alternating high and low voltage levels may be symbolically represented by 0s and 1s. This is the simplest way to represent a binary string (bit-string).

Each 0 or 1 is called a **bit**. Various codes combine bits to represent information stored in a computer.

---

**Analog signals**

PCs often communicate via modems over telephone lines using analog signals which are formed by continuously varying voltage levels.

---

**How signals travel?**

There are three types of transmission media, each with many variations:

- **Conductive metal**, like copper or iron, that carries both digital and analog signals; coaxial cable and twisted wire pairs are examples,
- **Transparent glass strand or optical fibre** that transmits data using light waves,
- **No physical connection** that transmits data using electromagnetic waves (as those used in TV or radio broadcast).

---

**How is information coded?**

Whether the medium uses light, electricity, or microwaves, we must answer perhaps the most basic of all communication questions:

**How is information coded in a format suitable for transmission?**
Bits

Regardless of implementation, all switches are in one of two states: open or closed, symbolically, 0 and 1.

Bits can store only two distinct pieces of information. Grouping them, allows for many combinations:
- two bits allow $2^2 = 4$ unique combinations: 00, 01, 10, 11
- three bits allow for $2^3 = 8$ combinations,
- ten bits allow for $2^{10} = 1,024$ combinations,
- fifty bits allow for $2^{50} = 1,125,899,068,426,524$ combinations,
- $n$ bits allow for $2^n$ combinations.

From bits to codes

Grouping bits allows one to associate certain combinations with specific items such as characters, numbers, pictures. Loosely speaking, this association is called a code. Not every association is a code as we shall soon learn.

A difficult problem in communications is to establish communications between devices that operate with different codes. There are standards, but not all standards are compatible!

The nice thing about standards is that you have so many to choose from.
– Tanenbaum, Computer Networks (2nd Ed.), 1988

Early codes: Morse

Originally created for Morse’s electric telegraph in 1838, by the American inventor Samuel Morse, the Morse code was also extensively used for early radio communication beginning in the 1890s.

The telegraph required a human operator at each end. The sender would tap out messages in Morse code which would be transmitted down the telegraph wire to a human decoder translating them back into ordinary characters.
Morse code is a variable-length code:
- letter codes have different lengths; the letter E code is a single dot (1000), the letter H code has four dots (1010101000);
- the code (0000000) for an inter-word gap (the ‘space’ character) is of length 7;

Reason: more frequent letters are assigned shorter codes, so messages can be sent quickly.

The Baudot code—also known as International Telegraph Alphabet No 2 (ITA2)—is named after its French inventor Émile Baudot. ITA2 is a fix-length code using 5 bits for each character (digits and letters). This code was developed around 1874.

With 5-bit codes we can name $2^5 = 32$ different objects, but we have 36 letters and digits (plus special characters) to code!

For example, the letter Q and digit 1 have the same code: 10111. In fact each digit’s code duplicates that of some letter.

Do you think we have got a problem?
More precisely, how can we tell a digit from a letter?

Answer: using the same principle that allows a keyboard key to represent two different characters. On the keyboard we use the Shift key; the Baudot code uses the extra information

11111 (shift down) and 11011 (shift up)

to determine how to interpret a 5-bit code. Upon receiving a shift down, the receiver decodes all codes as letters till a shift up is received, and so on.
Data Transmission  Codes  Analog and Digital Signals  Compression  Data Integrity  Powerline communications

Early codes: Baudot code

Here is an example. ABC123, is coded from left to right as follows:

1111 0001 1101 0110 1101 1011 1001 0001

Early codes: BCD, BCDIC, ASCII codes

- BCD stands for binary-coded decimal, a code developed by IBM for its mainframe computers using 6-bit codes;
- BCDIC stands for binary-coded decimal interchange code, an expansion of BCD including codes also for non-numeric data;
- ASCII (pronounced [ˈæski]) stands for the American Standard Code for Information Interchange; it is a 7-bit code that assigns a unique combination to every keyboard character and to some special functions.

ASCII code (decimal, binary, hexadecimal)

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>Bin</th>
<th>Hex</th>
<th>Oct</th>
<th>Bin</th>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>1</td>
<td>0</td>
<td>001</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>001</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>2</td>
<td>0</td>
<td>010</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>010</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>3</td>
<td>0</td>
<td>011</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>011</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>4</td>
<td>0</td>
<td>100</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>100</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>5</td>
<td>0</td>
<td>101</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>101</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>6</td>
<td>0</td>
<td>110</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>110</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>7</td>
<td>0</td>
<td>111</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>111</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Each code corresponds to a printable or unprintable character. Printables characters include letters, digits, and special punctuation (commas, brackets, question marks).

Unprintable characters are special functions (e.g. line feed, tab, carriage return, BEL, DC1/XON/ctrl-Q, DC3/XOFF/ctrl-S).

Standard ASCII has 128 different characters.

Extended ASCII codes (e.g. ISO-8859-1, Mac OS Roman, ...) have an additional 128 characters.
What is a code?

A code is the assignment of a unique string of characters (a codeword) to each character in an alphabet.

A code in which the codewords contain only zeroes and ones is called a binary code.

The encoding of a string of characters from an alphabet (the cleartext) is the concatenation of the codewords corresponding to the characters of the cleartext, in order, from left to right. A code is uniquely decodable if the encoding of every possible cleartext using that code is unique.

For example, here are two possible binary codes for the alphabet \{a, c, j, l, p, s, v\}:

<table>
<thead>
<tr>
<th></th>
<th>code 1</th>
<th>code 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>c</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>j</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>l</td>
<td>0001</td>
<td>10</td>
</tr>
<tr>
<td>p</td>
<td>00001</td>
<td>0</td>
</tr>
<tr>
<td>s</td>
<td>000001</td>
<td>1</td>
</tr>
<tr>
<td>v</td>
<td>000001</td>
<td>101</td>
</tr>
</tbody>
</table>
Both code 1 and code 2 satisfy the definition of a code. However,
- code 1 is uniquely decodable, but
- code 2 is not uniquely decodable; for example, the encodings
  of the cleartexts “pascal” and “java” are both

\[001010101010\]

Sardinas-Patterson algorithm

The algorithm computes the sets \(C_i\) in increasing order of \(i\). As soon as one of the sets \(C_i\) contains a word from \(C\) or the empty word, then the algorithm terminates and answers that the given code is not uniquely decodable.

Otherwise, once a set \(C_i\) equals a previously encountered set \(C_j\) with \(j < i\), it answers that the given code is uniquely decodable.

Consider \(C = \{0, 1, 01\}\). Applying the Sardinas-Patterson algorithm we get

\[C_1 = \{1\}, \quad C \cap C_1 = \{1\},\]

so the code \(C\) is not uniquely decodable. Indeed, \(01 = 0 - 1\).

Applying the Sardinas-Patterson algorithm to \(C = \{0, 10, 01\}\) we get

\[C_1 = \{1\}, \quad C_2 \cap C = \{0\},\]

so the code \(C\) is not uniquely decodable: \(010 = 0 - 10 = 01 - 0\).
Sardinas-Patterson algorithm

Applying the Sardinas-Patterson algorithm to
\[ C = \{1, 011, 10110, 1110, 10011\} \]
we get
\[ C_1 = \{110, 0011, 10\}, \]
\[ C_2 = \{10, 011, 0\}, \text{ and } C_1 \cap C = \{011\}, \]
so the code \( C \) is not uniquely decodable. Indeed:
\[
011101110011 = 011101110011011 = 011101110011011 = 011101110011011011011011011011.
\]

Prefix codes

Every fixed-length code is a prefix code.

There can be no prefixes in the code table, because no codeword is any longer or shorter than any other.

Therefore, ASCII is a prefix code.

Given a sequence of lengths, can we construct a prefix code whose codewords have exactly those lengths?

Kraft’s theorem. A prefix code exists for codewords lengths \( l_1, l_2, \ldots, l_N \) if and only if
\[
2^{-l_1} + 2^{-l_2} + \ldots + 2^{-l_N} \leq 1.
\]

Arrange the lengths in increasing order so (after relabelling) \( l_1 \leq l_2 \leq \ldots \leq l_N \). Take as the first codeword, \( w_1 = 0^{l_1} \), i.e. 00...0 for \( l_1 \) times. If \( w_1, w_2, \ldots, w_i \) have been constructed, choose \( w_{i+1} \) to be the first string (lexicographically) of length \( l_{i+1} \) such that no \( w_j \) is a prefix of it. The above inequality guarantees that \( w_{i+1} \) always exists.
Prefix codes

Here is an example. Consider the lengths 3, 2, 4 which satisfy the condition in Kraft theorem:

\[ 2^{-3} + 2^{-2} + 2^{-4} = \frac{7}{16} < 1. \]

We arrange the lengths in increasing order 2, 3, 4, and we have:
- \( w_1 = 00 \)
- \( w_2 = \) first string in the set 000, 001, 010, 011, 100, 101, 110, 111 such that 00 is not its prefix = 010
- \( w_3 = \) first string in the set 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 such that its prefixes do not include both 00 and 010 = 0110

Kraft’s inequality

Kraft’s inequality (1)

\[ 2^{-h} + 2^{-b} + \ldots + 2^{-lw} \leq 1, \]

can be thought of in terms of a constrained budget to be spent on codewords, with shorter codewords being more expensive:

- If Kraft’s inequality holds with strict inequality, the code has some redundancy.
- If Kraft’s inequality holds with strict equality, the code in question is a complete code.
- If Kraft’s inequality does not hold, the code is not uniquely decodable.

Questions

1. How does symbolic data relate to electrical signals, microwaves, and light waves?
2. What does a 0 or a 1 actually look like as it travels through a wire, optical fibre, or space?
3. How many bits can a signal transmit per unit of time?
4. What effect does electrical interference (noise) have on data transmission?
Data Transmission Codes
Analog and Digital Signals
Compression Data integrity Powerline communications

Analog vs. digital signals

Encoding Bits in Analog Signals

Because digital signals can alternate between two constant values—say “high voltage” and “low voltage”—we simply associate 0 with one value and 1 with the other. The actual values are irrelevant.

**Non-Return to Zero (NRZ):** A 0 is transmitted by raising the voltage level to high, and a 1 is transmitted using a low voltage. Alternating between high and low voltage allows for the transmission of any string of 0s and 1s.

**Non-Return-to-Zero-Inverted Encoding (NRZI):** A 0 is encoded as no change in the level. However a 1 is encoded depending on the current state of the line. If the current state is 0 [low] the 1 will be encoded as a high, if the current state is 1 [high] the 1 will be encoded as a low. Used in USB.

Synchronisation Problem

What is being transmitted in NRZ? In NRZI? A string of 0s, in either case. But... how many?

NRZ and NRZI require a *clock signal* as well as a *data signal*. Sending the clock signal requires an additional connection with the same latency as the data signal, otherwise the clock will be skewed and the data will be decoded incorrectly.
The Manchester code uses signal changes to keep the sending/receiving devices synchronised. It encodes 0 and 1 by changing the voltage:

0 is represented by a change from high to low and 1 is represented by a change from low to high.

Note: the signal will never be held constant longer than a single bit interval, no matter what data is being transmitted.

A Manchester signal can change levels twice every $T$ seconds, and must change at least once. An NRZ or NRZI signal can change level at most once every $T$ seconds: this is half the bandwidth of a Manchester signal. (Are there more efficient encodings of a clock signal?)

The process of adding a data signal (e.g. a digital bitstream from a PC) to an analog carrier signal is called modulation. The process of extracting the data from a modulated signal is called demodulation.

Frequency modulation is used in FM radio transmission. The analog audio signal (typically limited to 15 kHz) modulates the analog carrier signal (e.g. 101.4 MHz for National FM in the Auckland region).

A modem, short for modulator/demodulator, is a device that does both conversions — for digital data and an analog carrier.

A sine wave is the simplest analog signal. There are three ways to adjust a sine wave:

1. changing its frequency,
2. changing its amplitude,
3. changing its phase.

See http://www.ltscotland.org.uk/5to14/resources/science/sound/index.asp.
Here are some numerical examples with reference to the previous picture:

In (a) the period for \( y = \sin(t) \) is \( p = 2\pi \),

In (b) the period for \( y = \sin(Nt) \) is \( p = 2\pi/N \),

In (b) the frequency is \( f = 1/p = N/2\pi \) Hz,

In (a), (b), and (d) the amplitude is \([-1, 1]\); and the peak amplitude is 1. In (c), the amplitude is \([-A, A]\) with a peak of \( A \).

A sine function can be written in the form:

\[
 s(t) = A\sin(2\pi ft + k),
\]

where \( s(t) \) is the instantaneous amplitude at time \( t \), \( A \) is the peak amplitude, \( f \) is the frequency, and \( k \) is the phase shift.

These three characteristics fully describe a sine function.
For electrical signals, the peak amplitude is measured in volts (V). The frequency is measured in hertz (Hz).

<table>
<thead>
<tr>
<th>Unit</th>
<th>Equivalent</th>
<th>Unit</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>second (s)</td>
<td>1 s</td>
<td>hertz (Hz)</td>
<td>1Hz</td>
</tr>
<tr>
<td>millisecond (ms)</td>
<td>$10^{-3}$ s</td>
<td>kilohertz (kHz)</td>
<td>$10^3$ Hz</td>
</tr>
<tr>
<td>microsecond (µs)</td>
<td>$10^{-6}$ s</td>
<td>megahertz (MHz)</td>
<td>$10^6$ Hz</td>
</tr>
<tr>
<td>nanosecond (ns)</td>
<td>$10^{-9}$ s</td>
<td>gigahertz (GHz)</td>
<td>$10^9$ Hz</td>
</tr>
<tr>
<td>picosecond (ps)</td>
<td>$10^{-12}$ s</td>
<td>terahertz (THz)</td>
<td>$10^{12}$ Hz</td>
</tr>
</tbody>
</table>

Sound can be transduced by a microphone, or by our ears, into an electrical (analog) signal. The audio signal from a microphone has the following relationships to our aural perceptions:

- the amplitude of the audio signal correlates with our perception of volume, and
- the frequency correlates with our perception of pitch.

A single sine function is not useful for data communication. We need to change one or more of its characteristics—amplitude, frequency and phase shift—to encode a signal. How do we decode a signal from a sine wave carrier that has been modulated by shifts in amplitude, frequency and/or phase?

Answer: Fourier Analysis. The French mathematician Jean Baptiste Fourier proved that any periodic function can be expressed as an infinite sum of sine and cosine functions of varying amplitudes, frequencies and phase shifts—a Fourier series.
Any periodic signal \( x(t) \) with period \( P \) can be expressed as a Fourier series:

\[
x(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty} \left[ a_i \cos \left( \frac{2\pi it}{P} \right) + b_i \sin \left( \frac{2\pi it}{P} \right) \right]
\]

The coefficients \( a_1, a_2, \ldots, b_1, b_2, \ldots \) are uniquely determined by this equation, and are called the Fourier transform of \( x(t) \).

For example, consider a unit-amplitude square wave \( s(t) \) with period \( 2\pi \):

\[
s(t) = \begin{cases} 
1, & \text{if } t \in [0, \pi) \cup [2\pi, 3\pi) \cup [4\pi, 5\pi) \cup \ldots, \\
-1, & \text{if } t \in [\pi, 2\pi) \cup [3\pi, 4\pi) \cup [5\pi, 6\pi) \cup \ldots,
\end{cases}
\]

Each term in this series can be expressed in our general form for sine waves:

\[
\frac{4}{i\pi} \sin(it) = A \sin(2\pi ft + k),
\]

where the amplitude, frequency and phase shift are

\[
A = \frac{4}{i\pi}, \quad f = \frac{1}{2\pi}, \quad k = 0.
\]
A square wave is thus a sum of sine functions with frequencies $f, 3f, 5f, \ldots$ and amplitudes $\frac{4}{\pi}, \frac{4}{3\pi}, \frac{4}{5\pi}, \ldots$.

\[ s(t) = \frac{4}{\pi} \sin(2\pi ft) + \frac{4}{3\pi} \sin[2\pi(3f)t] + \frac{4}{5\pi} \sin[2\pi(5f)t] + \cdots \]

The term with frequency $f$ is called the **fundamental frequency**; the term with frequency $3f$ is called the **third harmonic**; the term with frequency $5f$ is called the **fifth harmonic**; etc. These are all **odd harmonics**.

We can compute a finite number $n$ of Fourier coefficients efficiently, in $O(n \log n)$ floating-point multiplications and additions, using an algorithm called the **fast Fourier transform**.

Now add the third harmonic to the fundamental, and plot it.

\[ y = \sin(t) + \sin(3t)/3 \; \text{plot}(t,y); \]

The Fourier series expansion for a square-wave is made up of a sum of odd harmonics. We show this graphically using MATLAB®.

We start by forming a time vector running from 0 to 10 in steps of 0.1, and take the sine of all the points. Let’s plot this fundamental frequency.

\[ t = 0:.1:10; \]
\[ y = \sin(t); \]
\[ \text{plot}(t,y); \]


Now use the first, third, fifth, seventh, and ninth harmonics.

\[ y = \sin(t) + \sin(3t)/3 + \sin(5t)/5 + \sin(7t)/7 + \sin(9t)/9 \; \text{plot}(t,y); \]
For a finale, we will go from the fundamental to the 19th harmonic, creating vectors of successively more harmonics, and saving all intermediate steps as the rows of a matrix.

These vectors are plotted on the same figure to show the evolution of the square wave.

```matlab
for k=1:2:19
    x=x+sin(k*t)/k;
    y((k+1)/2,:) = x;
end
plot(y(1:2:9,:))
title('The building of a square wave')
```

The Gibbs phenomenon is an overshoot (or “ringing”) of Fourier series occurring at simple discontinuities (edges): we will never get “there” in finite time.

Here is a 3-D surface representing the gradual transformation of a sine wave into a square wave.

```matlab
surf(y); shading interp axis off ij
```

The fast Fourier transform can be used in digitised versions of analog signal-processing devices.

- **Filters** attenuate certain frequencies while allowing others to pass. The Bass control on a stereo system is an adjustable lowpass filter which limits the amount of low-frequency sound in the output. The Treble control is an adjustable highpass filter. Stereo equalisers have many adjustable bandpass filters.
- **Tuners** extract one modulated signal from a signal which has been modulated by many signals. Tuners are necessary in radio and TV receivers, to select one station or channel from all of the ones that are currently being broadcast.
The **bit rate** describes the information-carrying capacity of a digital channel, and is measured in bits per second (b/s).

The range of frequencies in a channel is called its **bandwidth**. Roughly:

* a higher-bandwidth channel has a higher bit rate.

(We will develop a more refined understanding, in a moment...)

The bit rate $R = f_s n$ is the product of the frequency $f_s$ at which symbols are sent, and the number of bits per symbol $n$.

Note that this equation is trivial, by a dimensional analysis:

$$\text{bits/second} = \frac{\text{bits} \cdot \text{symbol}}{\text{second} \cdot \text{symbol}}$$

$$= \frac{\text{bits}}{\text{symbol}} \cdot \frac{\text{symbols}}{\text{second}}$$

As a shorthand for symbols/second, we write baud – after Émile Baudot, the inventor of the Baudot code. The abbreviation is Bd, and its units are symbols per second.

**Nyquist theorem.** In a distortion-free transmission, the baud rate $f_s$ is at most twice the maximum frequency $f$ of the medium.

Since the baud rate $f_s$ is upper-bounded by $2f$, the bit rate $R$ is at most $2fn$ when each symbol carries $n$ bits.

Telephone connections have a maximum frequency $f$ of 3300 Hz.
Data Transmission Codes

Analog and Digital Signals

Compression

Data integrity

Powerline communications

Noisy channels

Many channels are noisy. Note the voltage difference between the first and last part of the transmitted signal.

![Diagram of noisy channels]

Sending data via signals

Noisy channels

If our SNR is reported as 25 dB, this is 2.5 bels, i.e. we have

\[
\text{SNR}_{\text{dB}} = \log_{10}(S/N) = 2.5 \text{ bels}
\]

This implies

\[
\frac{S}{N} = 10^{2.5}
\]

or

\[
S = 10^{2.5} \times N = 100\sqrt{10} \times N \approx 316N
\]

The American scientist Claude Shannon, inventor of the classical theory of information, refined Nyquist’s theorem by taking into account the channel’s noise:

**Shannon’s theorem.** In a noisy transmission,

\[
\text{bit rate (in b/s)} \leq \text{bandwidth (in Hz)} \times \log_2(1 + S/N)
\]

The quantity \(\log_2(1 + S/N)\) is the maximum number of bits that can be transmitted per cycle on this channel.

Note that decreasing the signal-to-noise ratio on any channel will decrease its information-carrying capacity. Somewhat surprisingly, a channel with any non-zero signal strength has some capacity (because its SNR must be greater than zero).
Noisy channels 5

Telephones carry audio frequencies in the range 300 Hz to 3300 Hz: this is a bandwidth of 3000 Hz. A good telephone connection has a signal-to-noise ratio of 35 dB. So,

\[ 3.5 \text{ bels} = \log_{10}(S/N) \text{ bels}, \]
\[ S = 10^{3.5} \times N \approx 3162N. \]

Using Shannon’s theorem we get:

\[ \text{bit rate} \leq \text{bandwidth} \times \log_2(1 + S/N) \]
\[ = 3000 \times \log_2(1 + 3162) \text{ b/s} \]
\[ \approx 3000 \times 11.63 \text{ b/s} \]
\[ \approx 34880 \text{ b/s} \]

Answer: Modern telephone systems use 64 kb/s digital signalling on their long-distance connections, and the downlink on a V.90 modem is able to interpret these digital signals when it receives data from your ISP. Your downlink is a channel with an 8 kbps signalling rate and 8 bits/symbol; the bandwidth of this channel is 4 kHz, not the 3 kHz of an analog voice connection. The uplink on a V.90 modem uses the 3 kHz analog voice channel, transmitting data at 33.6 kb/s if your phone line isn’t noisy, and transmitting at a lower bitrate if you have a noisy line.

V.92 modems can upload at 56 kb/s, but as far as I know, no NZ ISP provides a V.92 dialup service.

Digital to analog conversion 1

Computer data transmitted over telephone lines

Voice information transmitted through a digital connection, using a codec (coder/decoder)
There are three main ways to encode a digital signal as an analog signal, corresponding to the three parameters in a sine wave. We can modulate

1. by frequency, for example by **frequency shift keying (FSK)**,
2. by amplitude, for example by **amplitude shift keying (ASK)**,
3. by phase modulation, for example **phase shift keying (PSK)**.

We can also use combined methods, such as **quadrature amplitude modulation (QAM)** which modulates both amplitude and phase.

**FSK** or **frequency modulation (FM)** assigns a digital 0 to one analog frequency and a 1 to another.

FSK, at one bit per symbol.

Note: we might use \( n \)-bit symbols, where each symbol is assigned one frequency in a set of \( 2^n \) frequencies.

**ASK** or **amplitude modulation (AM)** assigns a digital ‘0’ to one analog amplitude, a ‘1’ to another amplitude, ... and (for AM at \( n \) bits per symbol) a ‘\( 2^n - 1 \)’ to yet-another amplitude.

ASK, at two bits per symbol.

**PSK** or **phase modulation (PM)** assigns each bit-string of a fixed length to one analog phase shift. **Quadrature amplitude modulation (QAM)** is a combination of amplitude and phase shift.

QAM with two amplitudes, four phases, and three bits per symbol.
Analog to digital conversion

In principle, any digital to analog encoding can be decoded. However some analog-to-digital decodings are easier than others.

In the most obvious analog-to-digital decoding, we start by sampling an analog signal at regular intervals. When our samples are analog, we are modifying the analog signal by “flattening” all its high-frequency components. This modification process is called pulse amplitude modulation (PAM).

PAM signals may seem digital, but they have analog amplitudes. In pulse code modulation (PCM), we define a set of $2^n$ amplitudes, where $n$ is the number of bits per symbol. The process of “rounding” an analog amplitude to its nearest available (“digital”) approximation is called quantisation.

Digital telephone signalling (DS0, E0, J0) in North America, Europe, and Japan, is PCM with 8 bits per symbol at 8 kbaud.

Why compression? 1

Digital media, by using sophisticated compression algorithms, can have significant performance benefits over analog media.

Here is an example. In the (obsolescent) PAL broadcast standard for television, the bandwidth is approximately 7 MHz. The SNR is roughly 20 dB, depending greatly on the location, design, and orientation of the antenna.

If we sample a PAL signal at the Nyquist rate of twice its bandwidth (14 MBd), and if we use the trivial digital encoding of 8 bits/sample for three channels (Red, Green, and Blue), then we will have a data rate of $3 \cdot 8 \cdot 14 = 336$ Mb/s, or 42 MB/s.

If we record all of these bytes for two hours ($= 2 \cdot 60 \cdot 60 = 7200$ seconds), we will have 302400 MB, or 302.4 GB. A digital video disk (DVD-5) can hold about 4.7 GB ...

Why compression? 2

A trivial form of data compression, for any analog signal, is to cut its bandwidth – we can use a low-pass filter to limit its high-frequency information. Analog PAL broadcasting, and VHS tapes, use this technique to compress the chrominance (colour) information in studio-quality TV recordings. The broadcast chroma is only about 2 MHz at 20 dB.

We could digitally sample the chroma signal (using PCM) at a rate of $4 \log_2(1 + 10^{20/2}) = 27$ Mb/s $\approx 3$ MB/s. The luminance signal is about 9 MB/s, for a total of 12 MB/s.

This is more than a 3:1 compression of our naive 42 MB/s encoding. However we need to get a 40:1 compression, down to about 1 MB/s, in order to record 90 minutes of video on a DVD.
How do you reduce bits and still keep enough information?

Guiding principle of compression: Discard any information (such as high-frequency chroma) that is unimportant; retain any information that is essential to the “meaning”.

Here is a simple example. Assume that you wish to email a large file consisting entirely of strings of capital letters. If the file has $n$ characters each stored as an 8-bit extended ASCII code, then we need $8n$ bits.

However, we don’t need all ASCII codes to code strings of capital letters: they use only 26 characters. We can make our own code with only 5-bit codewords ($2^5 = 32 > 26$), code the file using this coding scheme, send the encoded file via email, and finally decode it at the other end. Big deal?

The size has decreased by $8n - 5n = 3n$, i.e. a 37.5% reduction.

Lossless and lossy compression

There are two different ways of data compression algorithms: lossless or lossy.

A lossless technique restores identically the compressed data file to the original. This is absolutely necessary for many types of data, for example: executable code, word processing files, tabulated numbers, etc. One cannot afford to misplace even a single bit of this type of information. A lossless technique (such as used in the GIF format), will compress at about one-half the original size.

Compressing by frequency

Compression techniques that allow a small amount of degradation are called lossy. Lossy techniques are much more effective at compression than lossless methods: the higher the compression ratio, the more noise added to the data. An image compressed with the lossy JPEG technique has about 1/12 of the original size.

JPEG is the best choice for digitised photographs, while GIF is used with drawn images, such as company logos that have large areas of a single colour.
Compressing by frequency


![Histogram of ASCII values.](image)

More than 96% of this file consists of only 31 characters: the lower case letters, the space, the comma, the period, and the carriage return.

This "pattern" can be exploited for compression: assign each of these 31 common characters a five-bit binary code:

- `00000 = a, 00001 = b, ..., 11110 = carriage return`

This allows 96% of the file to be reduced in size by 5/8.

Huffman code

The **Huffman code** exploits frequency to the extreme. For illustration, assume that we have five characters, A-E, whose frequencies are as follows:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
</tr>
<tr>
<td>E</td>
<td>30</td>
</tr>
</tbody>
</table>

The average number of bits required per original character is:

$$0.96 \times 5/100 + 0.04 \times 13 = 5.32,$$

showing an overall compression ratio of 8 bits/5.32 bits ≈ 1.5:1.
To construct the Huffman codeword for the string we follow the following algorithm:

1. To each character we associate a binary tree consisting of just one node. To each tree we assign the tree’s weight. Initially, the weight of nodes is exactly its frequency: 0.25 for A, 0.15 for B, etc.

2. Calculate the two lightest-weight trees (choose any if there are more than two). Merge the two chosen trees into a single tree with a new root node whose left and right sub-trees are the two we chose. The weight of the new tree is the sum of the weights of the merged trees.

3. Repeat the procedure till one tree is left.

When completed, each of the original nodes is a leaf in the final tree. Arcs are labelled with 0 and 1 as follows: we assign a 0 each time a left child pointer is followed and a 1 for each right child pointer.

As with any binary tree, there is a unique path from the root to any leaf. The path to any leaf defines the Huffman code of the character on labelling the leaf.

Let us apply this procedure to the character strings A, B, C, D, E.
As one can see, the algorithm for constructing the Huffman tree is very “tolerant”, it gives a lot of flexibility. In the above example one could, for instance, choose putting D on the right-hand side of the merged tree in stage (c) and swap A and E in stage (d). The result will be the Huffman code:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency (%)</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
<td>00</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>100</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>30</td>
<td>01</td>
</tr>
</tbody>
</table>

Huffman code properties:

- There are many Huffman codes which can be associated to the same characters and weights. So, we should better speak of a Huffman code instead of the Huffman code.
- All Huffman codes are, in general, variable-length, frequency-dependent, prefix codes. Consequently, Huffman codes are uniquely decodable.

Let us decode the codeword 01110001110110110111:

Huffman codes are useful only in case we know the frequency of characters. Many items that travel the communications media, including binary files, fax data, and video signals, do not fall into this category.

**Run-length encoding** analyses bit-strings by looking for long runs of 0 or 1. Instead of sending all bits, it sends only how many of them are in the run. Fax transmission is well-served by this technique as potentially a large part of a page is white space, corresponding to a long run of 0s.
Run-length encoding

The sender transmits only the length of each run as a fixed-length integer; the receiver gets each length and generates the proper number of bits in the run, inserting the other bit in between. Here is an example:

```
bit-stream 0 ... 010 ... 010 ... 010 ... 010 ... 0 00 bits
number of Os in run  14  9  20  30  11
(a) Stream prior to compression
run lengths (binary) 1110 1001 0000 1111 0111 1111 0000 0000 0111 40 bits
run lengths (decimal) 14 9 0 15 5 15 0 0 11
(b) Run-length-encoded stream
```

This technique works well when there are many long 0 runs. Note that the symbol "15" indicates a run of length 15 or more, so the run of thirty 0s is encoded as “15”, “15”, “0”.

Relative encoding

A single video image may contain little repetition, but there is a lot of repetition over several images.

Reason: a) a USA TV signal sends 30 pictures per second, b) each picture generally varies only slightly from the previous one.

The relative encoding or differential encoding works as follows:
- the first picture is sent and stored in a receiver’s buffer;
- the second picture is compared with the first one and the encoding of differences are sent in a frame format; the receiver gets the frame and applies the differences to create the second picture;
- the second picture is stored in a receiver’s buffer and the process continues.

Here is an example of relative encoding:

```
<table>
<thead>
<tr>
<th>First frame</th>
<th>Second frame</th>
<th>Third frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>5762866356</td>
<td>5762866356</td>
<td>5762866356</td>
</tr>
<tr>
<td>6575563247</td>
<td>6576563237</td>
<td>6586563337</td>
</tr>
<tr>
<td>8468564885</td>
<td>8468564885</td>
<td>8468564885</td>
</tr>
<tr>
<td>5129865566</td>
<td>5139865576</td>
<td>5139765586</td>
</tr>
<tr>
<td>5529968951</td>
<td>5529968951</td>
<td>5529968951</td>
</tr>
</tbody>
</table>
```

Transmitted frame contains the encoded differences between the first and second frames. Transmitted frame contains the encoded differences between the second and third frames.
The **Lempel-Ziv compression** method looks ahead in the input, finding the longest previously-transmitted string which is a prefix of the input. It transmits a reference to that previous string, thereby avoiding sending the same string more than once.

The Lempel-Ziv compression is widely used as it is simple and compresses roughly by one-half:

- UNIX `compress` command,
- `gzip` on UNIX,
- V.42bis compression standard for modems,
- GIF (Graphics Interchange Format).

**JPEG**

JPEG is an acronym for the Joint Photographic Experts Group and JPEG compression is a **lossy** data compression method. This means that the image obtained after decompression may not coincide with the original image. Lossy compression is acceptable for images because of the inherent limitations of the human optical system.

There are three phases in a JPEG compression:

- the discrete cosine transform (DCT),
- quantisation,
- encoding phase.
In the DCT phase, each component of the image is “tiled” into sections of 8 × 8 pixels each; dummy data fills incomplete blocks.

Usually, there are three components in a JPEG image, corresponding to the (Y', U, V) we discussed earlier. Other colour-spaces and grey-scale (a single 8-bit component) may be used.

The chroma components are usually downsampled, i.e. encoded at lower spatial resolution than the luma.

Then, each tile in each component is converted to frequency space using a two-dimensional forward “discrete cosine transform”, using the basis functions shown below.

The human eye can see small differences in brightness over a relatively large area, but is insensitive to brightness variation at high frequency.

JPEG attenuates the high frequency signal components by dividing each component $G_{ij}$ in the frequency domain by a constant $Q_{ij}$, where the value of the constant is larger for the higher frequency components (i.e. the ones with larger $i,j$ values). After division, the component is rounded to the nearest integer. This is the *quantisation phase*, in which the main *lossy* operation takes place.

As a result, in the *encoding* phase, many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers; they take many fewer bits to store.

The compression ratio of JPEG depends greatly on the divisors used during the quantisation phase. These are controlled by a “Quality” parameter.

At 10:1 compression, it is difficult to distinguish the compressed image from the original one.

At 100:1 compression, a JPEG-compressed image is usually still recognisable but has many visual artifacts, especially near sharp edges.
GIF

**GIF (Graphics Interchange Format)** compresses by: a) reducing the number of colours to 256, and b) trying to cover the range of colours in an image as closely as possible.

It replaces each 24-bit pixel value with an 8-bit index to a table entry containing the colour that matches the original “most closely”. In the end, a variation of the Lempel-Ziv encoding is applied to the resulting bit values.

GIF files are lossy if the number of colours exceeds 256 and lossless otherwise.

MPEG and MP3

The **Moving Picture Experts Group (MPEG)** is a working group of ISO/IEC charged with the development of video and audio encoding standards.

The video codecs from MPEG use the discrete cosine transform techniques of the JPEG image compressor, and they also take advantage of redundancy between successive frames of video for “inter-frame compression”. Differences between successive frames can be encoded very compactly, when the motion-prediction techniques are successful.

Note: MPEG holds patents, and charges license fees. In June 2002, China formed a working group to develop an alternative set of audio and video codecs. The group was successful, see [http://www.avs.org.cn](http://www.avs.org.cn), however the H.263 (MPEG-2) and H.264 (MPEG-4 Part 10) codecs are still commonly used in China.

MPEG has standardised the following compression formats and ancillary standards:

- MPEG-1: video and audio compression standard; it includes the popular Layer 3 (MP3) audio compression format
- MPEG-2: transport, video and audio standards for broadcast-quality television
- MPEG-4: expands MPEG-1 to support video/audio “objects”, 3D content, low bit-rate encoding and support for Digital Rights Management

**MPEG-1 Audio Layer 3**, better known as **MP3**, is a popular digital audio encoding, lossy compression format, and algorithm.

MP3 is based on the **psycho-acoustic model**, **auditory mask**, and **filter bank**.

Psycho-acoustics is the study of the human auditory system to learn what we can hear and what sounds we can distinguish. In general we can hear sounds in the 20 Hz to 20 kHz range, but sounds with close frequencies (for example, 2000 Hz and 2001 Hz) cannot be distinguished.
The auditory mask is the following phenomenon: if a sound with a certain frequency is strong, then we may be unable to hear a weaker sound with a similar frequency.

The filter bank is a collection of filters, each of which creates a stream representing signal components of a specified range. There is one filter for each of the many frequency ranges. Together they decompose the signal into sub-bands.

The MPEG standards define audio and video codecs, as well as file formats.

The MP4 file format (MPEG-4 Part 14) is based on Apple’s QuickTime container format. The file header indicates what codecs are used for video and audio, and it also gives values for important parameters such as the video frame rate, horizontal and vertical resolution (i.e. number of pixels per line, and number of lines per frame), and colourspace.

The video codecs defined in MPEG-4 are all extensions of JPEG (or closely-related DCT-based techniques). Each video frame can be individually JPEG-encoded, however usually each JPEG-encoded frame (an “I-frame”) is followed by a few frames which have been encoded, much more compactly, using codewords which refer to 8x8 image blocks in the preceding (or following) I-frame.

These video compressors are especially successful in video with stationary backgrounds, but are not as effective in scenes with a lot of differently-moving objects – such as the leaves on a tree, on a windy day. When compression ratios are high for video with high-frequency information, JPEG-like artifacts become apparent e.g. sharp edges are sometimes rendered at very low resolution (as 8x8 blocks).
Why check integrity?

Data can be corrupted during transmission—many factors can alter or even wipe out parts of data.

**Reliable systems must have mechanisms for detecting and correcting errors.**

The capability to detect when a transmission has been changed is called **error detection**. In some cases a message with errors is discarded and sent again, but not always this is possible (e.g. in real-time viewing). In the later case the error has to be fixed (in real-time) and the mechanism doing this job is called **error correction**.

Types of errors

There are two types of errors:

1. **single-bit error**, when only one bit in the data has changed,
2. **burst error**, when two or more bits in the data have changed.

Single-error bit

Burst error
One way to check errors is by sending every data unit twice. A bit-by-bit comparison between the two versions is likely to identify all errors. The system is reasonably accurate (assuming that the errors are randomly distributed), but inefficient. Transmission time is doubled, and storage costs have increased significantly (because the transmitter and receiver must both retain a complete copy of the message).

A better approach is to add a fixed extra information to each segment (packet) of the message—this technique is called redundancy. There is no additional information in these error-checking bits, so they can be discarded after the check is completed. Note: natural languages are redundant, for example it is possible to understand most English sentences if vowels are omitted: FCTSSTRNGRTHNFLCTN.

There are three main types of redundancy checks:

1. parity check,
2. checksums,
3. cyclic redundancy check (CRC).

Low-density parity-check (LDPC) codes were the first to allow data transmission rates close to the theoretical maximum, the Shannon Limit.

A redundant bit, called **parity bit**, is added to every data unit so that the total number of 1s in the unit (including the parity bit) becomes even (or odd).
A simple parity check will detect all single-bit errors.
It can detect multiple-bit errors only if the total number of errors is odd.

Question: If the bit error rate (BER) is 0.1%, and errors are equiprobable at each bit in a 999-bit message with a single parity bit, what is the probability of an undetected error?

Answer: use a little probability theory. Calculate
- the probability of no error: \((99.9\%)^{1000} \approx 37\%\),
- the probability of a single error as \(1000(0.1\%)(99.9\%)^{999} \approx 37\%\),
- the probability of a double error: \({1000 \choose 2}(0.1\%)^2(99.9\%)^{998} \approx 18\%\),
- the probability of a triple error: \({1000 \choose 3}(0.1\%)^3(99.9\%)^{997} \approx 6\%\),
- the probability of a quadruple error: 2%,
- and the probability of more than four errors as less than 1%.

The chance of an undetected error is thus about 20% for 999-bit messages with a single parity bit, on a channel with a BER of 0.1%.

Would you be happy with a digital communication system that introduces errors in 20% of your messages?

In a **two-dimensional parity check** a block of bits is organised in a table and parity is checked on both dimensions.

- First one calculates the parity bit for each data unit.
- Second one calculates the parity bit for each column and one creates a new row of 8 bits—the parity bits for the whole block.

A two-dimensional parity check significantly increases the likelihood that a burst error will be detected.

*Definition.* If a message \(m_1m_2\ldots m_n\) contains a single **burst error** of length \(B\), then its bit-errors are confined to a single subsequence \(m_i\ldots m_{i+B-1}\) of length \(B\).
The method divides all data bits into 32-bit groups and treats each as an integer value. The sum of all these values gives the checksum. Any overflow that requires more than 32 bits is ignored.

An extra 32 bits representing in binary the checksum is then appended to the data before transmission.

The receiver divides the data bits into 32-bit groups and performs the same calculation. If the calculated checksum is different to the value received in the checksum field, then an error has occurred.

A checksum is more sensitive to errors than a single-bit parity code, but it does not detect all possible errors. For example, a checksum is insensitive to any errors which *simultaneously* increases (by any value \( c \)) one of the 32-bit groups while also decreasing another 32-bit group by the same value \( c \).

The most powerful (and elaborate) redundancy checking technique is the cyclic redundancy check (CRC).

CRC is based on binary division: a string of redundant bits—called the CRC or the CRC remainder—is appended to the end of the data unit such that the resulting data unit is exactly divisible by a second, predetermined binary number.

The receiver divides the incoming data by the same number. If the remainder is zero, the unit of data is accepted; otherwise it is rejected.
Error correction

Error-correcting codes include so much redundant information with the unit block that the receiver is able to deduce, with high likelihood, not only how many bits are in error, but also which bits are incorrect. After these bits are inverted, all errors are corrected in the received message.

These codes are used when the error rate is high (e.g. on a WiFi channel). Detecting an error and then re-transmitting the message is very inefficient on a noisy channel, because every retransmission is likely to have errors.

Note that we cannot absolutely guarantee error correction, unless (somehow) we can place an upper bound on the total number of bit-errors in a message. In particular, if all of the error-check bits may be simultaneously in error, then any error in the data bits might not be corrected—and it may not even be detected.

Forward error correction

We examine the simplest case, i.e. single-bit errors. A single additional bit can detect single-bit errors. Is it enough for correction?

To correct a single-bit error in an ASCII character we must determine which of the 7 bits has changed. There are eight possible situations: no error, error on the first bit, error on the second bit, ..., error on the seventh bit. Apparently we need three bits to code the above eight cases. But what if an error occurs in the redundancy bits themselves?

Clearly, the number of redundancy (or, error control) bits required to correct \( n \) bits of data cannot be constant, it depends on \( n \).

To calculate the number of redundancy bits \( r \) required to correct a \( n \) bits of data we note that:

- with \( n \) bits of data and \( r \) bits of redundancy we get a code of length \( n + r \).
- \( r \) must be able to handle at least \( n + r + 1 \) different states, one for no error, \( n + r \) for each possible position,
- therefore \( 2^r \geq n + r + 1 \).
Forward error correction

In fact we can choose \( r \) to be the smallest integer such that \( 2^r \geq n + r + 1 \). Here are some examples:

<table>
<thead>
<tr>
<th>Number of data bits ((n))</th>
<th>Minimum number of redundancy bits ((r))</th>
<th>Total bits ((n + 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>

The Hamming code

In a Hamming code for bit-strings of length \( 2^n - 1 \), each \( r \) bit is the parity bit for a specific combination of data bits, where the combinations follow a binary pattern shown below:

- \( r_1 \): parity on bits 1, 3, 5, 7, 9, 11, 13, 15, \ldots, \( 2^n - 1 \). That is, check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc.
- \( r_2 \): parity on bits 2, 3, 6, 7, 10, 11, 14, 15, \ldots. That is, skip 1 bit, check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc.
- \( r_4 \): parity on bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, \ldots. That is, skip 3 bits, check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc.
- \( r_2^n - 1 \): parity on bits \( 2^{n-1} \) through \( 2^n - 1 \). That is, skip \( 2^{n-1} - 1 \) bits, and check \( 2^{n-1} \) bits.

The Hamming code is a practical solution which detects and corrects all single-bit errors in data units of any length.

A Hamming code for 7-bit ASCII code has 4 redundancy bits. These bits can be added in arbitrary positions, but it is simplest if we do this on positions 1, 2, 4, and 8. Data bits are marked with \( d \) and parity bits are denoted by \( r_8, r_4, r_2, r_1 \).
This picture describes the calculation of $r$ bits for the ASCII value of 1001101: start with data and repeatedly calculate, one by one, the parity bits $r_1$, $r_2$, $r_4$, $r_8$:

Imagine that instead of the string 10011100101 the string 10010100101 was received (the 7th bit has been changed from 1 to 0).

The receiver recalculates 4 new parity bits using the same sets of bits used by the sender plus the relevant parity $r$ bit for each set. Then it assembles the new parity values into a binary number in the order used by the sender, $r_8$, $r_4$, $r_2$, $r_1$. This gives the location of the error bit. The sender can now reverse the value of the corrupted bit!

The basic Hamming code cannot correct multiple-bit errors, but can easily be adapted to cover the case where bit-errors occur in bursts.

Burst errors are common in satellite transmissions, due to sunspots and other transient phenomena which, occasionally, greatly increase the error rate for a brief period of time.

Scratches on CDs and DVDs introduce burst errors into the data read from these disks.
To protect a message of length $N$ against a single burst error of length $B \leq C$, we can break it up into groups (columns) of length $C$, then transmit it in transposed order with a Hamming code for each row.

No burst error of length $B < C$ can introduce more than one bit error in a row, so the row-wise Hamming codes are sufficient to correct a single burst error.

On the next slide, we show an example of the technique with $N = 36$ data bits. These bits are organised into columns of length $C = 6$, where the 6 data bits in each row are protected by a 5-bit Hamming code. A burst error of length 5 has affected two columns in our example, but has introduced at most one error in each row – so this error can be corrected by the Hamming code for the affected rows.

Hamming codes are pretty easy to understand, and they are pretty easy to implement – but they are not an effective way to protect very long messages against errors in arbitrary positions (i.e. multiple bit errors that aren’t in bursts).

Much more powerful error-correcting codes were developed by Bose, Chaudhari, and Hocquenghem in 1959-60. These are the BCH codes. The Reed-Solomon codes (also developed in 1960) are a very important sub-class of the BCH codes. Efficient decoding algorithms, suitable for special-purpose hardware implementation, were developed in 1969; these were used in satellite communications.

Nowadays, Reed-Solomon error-correction is used routinely in compact disks and DVDs.

A Hamming code uses $r = \log_2 n$ bits to guard $n$ data bits, so we can send only $2^r$ different messages in a Hamming-protected message of length $n + r$.

This might, or might not, be the appropriate ratio of error-checking (redundant) symbols to information-carrying symbols for this channel. Let’s develop a little more theory...
The Hamming distance between two $m$-bit strings is the number of bits on which the two strings differ.

Given a finite set of codewords one can compute its minimum distance, the smallest value of the distance between two codewords in the set.

If $d$ is the minimum distance of a finite set of codewords, then the method can detect any error affecting fewer than $d$ bits (such a change would create an invalid codeword) and correct any error affecting fewer than $d/2$ bits.

For example, if $d = 10$, and 4 bits of a codeword were damaged, then the string cannot possibly be a valid codeword: at least 10 bits must be changed to create another valid codeword.

If the receiver assumes that any error will affect fewer than 5 bits, then she needs only to find the closest valid codeword to the received damaged one to conclude that it is the correct codeword. Indeed, any other codeword would have had at least 6 bits damaged to resemble the received string.

When designing a code that can correct $d/2$ or fewer errors in a message, we must select codewords which are at least Hamming distance $d$ from each other. It's an interesting combinatorial puzzle... randomly-chosen codebooks do pretty well (but would require decoders to use large – and therefore slow and/or expensive – lookup tables)...

Error correction on the Internet is performed at multiple levels

- Each Ethernet frame carries a CRC-32 checksum. The receiver discards frames if their checksums do not match.
- The IPv4 header contains a header checksum of the contents of the header (excluding the checksum field). Packets with checksums that don’t match are discarded.
- The checksum was omitted from the IPv6 header, because most current link layer protocols have error detection.
- UDP has an optional checksum. Packets with wrong checksums are discarded.
- TCP has a checksum of the payload, TCP header (excluding the checksum field) and source- and destination addresses of the IP header. Packets found to have incorrect checksums are discarded and will eventually be retransmitted (when the sender receives three identical ACKs, or times out).

[Started 19 years ago DEF CON is a 15,000-person, four-day convention where anyone with $150—in cash only—can learn the latest tricks and trade of computer hacking, lock picking and security breaching. The following letter comes from http://tinyurl.com/3zc3mfc.]

Hi John,
Great talking with you!
You are about to enter one the most hostile environments in the world. Here are some safety tips to keep in mind ...
- Your hotel key card can be scanned by touch, so keep it deep in your wallet.
- Do not use the ATM machines anywhere near either conference. Bring cash and a low balance credit card with just enough to get you through the week.
DEF CON, Las Vegas, Nevada

- Turn off Fire Sharing, Bluetooth and Wi-Fi on all devices. Don’t use the Wi-Fi network unless you are a security expert; we have wired lines for you to use.
- Don’t accept gifts, unless you know the person very well - a USB device for instance.
- Make sure you have strong passwords on ALL your devices. Don’t send passwords “in the clear,” make sure they are encrypted. Change your passwords immediately after leaving Vegas.
- Don’t leave a device out of sight, even for a moment.
- People are watching you at all times, especially if you are new to the scene.
- Talk quietly. Conduct confidential phone calls off site ...
That is it for now.

What is powerline communications?

Powerline Communications can provide:
- High speed digital Internet.
- The Ascom powerline adapter telephony (fax included).
- Smart homes (remote maintenance and in-house control of internet enabled household appliances like refrigerators, heating systems, smoke and fire alarm systems).
- Monitored security.
- Health care services (monitor children and people in need from any internet connection).
- Utilities (online reading of utility meters for easy up-to-date billing).

A model of PLC (after Sankar, 2004)

picture source: Phil. Trans. R. Soc. A 364 (2006), 3200
**Why powerline communications?**

- No more wires, just plug in.
- Share your Internet connection.
- Mobility for computers and appliances.
- Easy to install.
- High transmission rate, right now 3 Mb/s (up to 30 times faster than ISDN) in uploading and downloading with speeds of 100 Mb/s possible.
- Secure data-encryption.
- Utilises existing power source for all communications.
- No more “urban/rural” divide.

**Why not switch NOW?**

One could, but there is still much work to be done before the technology is widely adopted.


A major technological problem is **noise**.

**Types of noise**

For TV and the internet, the main problem is the “white” background noise, which randomly corrupts individual bits. Codes have been constructed to deal with white noise. [They don’t always work: digital TV locks up in bad weather, for example.]

In power lines we may have “white” background noise, but also

- permanent narrow-band noise which affects some frequency over a period of time,
- impulse noise, that is short duration noise which affects many/all time slots.

Classical error-correcting coding does not work for PLC because of interference. Fortunately . . .

**Permutation codes**

We have seen that a signal, a sine function, is determined by frequency, amplitude, and phase. Radio signals can be transmitted in two ways: AM (amplitude modulation) and FM (frequency modulation).

With AM the signal has a fixed frequency and its amplitude is varied to represent the sounds. With FM you fix the amplitude and vary the frequency. The advantage of FM over AM is that the signal is not greatly affected by noise at a single frequency. This physical property can be used to design one type of error-correcting codes for PLC: a block of bits is represented by a series of pulses of various frequencies chosen from a finite set.
Permutation codes 2

Consider a five-letter alphabet \{A, B, C, D, E\} and design the permutation code which represents each letter as a sequence of five frequencies, transmitted in turn in five successive time slots.

![Frequency charts for A to E](image)

picture source: New Scientist, 24 March 2007, 51

Permutation codes 3

The above code allows its user to detect whether there are "lots of noise": the code can correct up to four errors in each sequence. Let us examine this claim in details.

If there is no noise when the letter A is transmitted, then the receiver will detect pulses of frequencies 1, 2, 3, 4, and 5.

![Frequency chart for A](image)

Permutation codes 4

Assume the channel suffers from interference. For example, assume that the signals on frequencies 3, 4 and 5 are distorted. The received signal, in successive time slots, will be of the following forms:

- 1 or 3 or 4 or 5
- 2 or 3 or 4 or 5
- 3 or 4 or 5
- 3 or 4 or 5
- 3 or 4 or 5

![Frequency charts for A](image)

Permutation codes 5

This agrees with 12345 (A) in all five time slots.

It disagrees with 23451 (B) in the first and last slots; with 34512 (C) in the last two slots; with 45123 (D) in the third and fourth slots; and with 51234 (E) in the second and third slots.

So, in spite of being distorted, the signal must be A, and not any of the other letters: the signal is decoded correctly.
If there is a sharp “bang!” in the fourth time slot that produces all five frequencies, the received signal will be:

1 or 3 or 4 or 5
2 or 3 or 4 or 5
3 or 4 or 5
1 or 2 or 3 or 4 or 5
3 or 4 or 5

This agrees with 12345 (A) in all five slots. It disagrees with 23451 (B) in the first and last slots; with 34512 (C) in the last slot; with 45123 (D) in the third slot; and with 51234 (E) in the second and third slots. The best agreement helps to decode the signal correctly.

Where next?

- Effective coding for PLC (mathematics).
- Understanding the “nature” of power line noise (engineering).
- Devising new Powerline Products (engineering)