Computer Networks: the Physical Layer

Norman Matloff
Dept. of Computer Science
University of California at Davis

September 27, 2005

Contents

1 Goals of This Unit

In this unit we will look at the physical properties of media which are used in computer networks. Our main emphasis is on the capacity of a network, meaning the maximum number of bits per second which can be loaded onto it. We will use the mathematical technique of **Fourier series** to illustrate the role of **bandwidth** in our network and in our data.

This will help not only to understand the physical limitations of networks, but also to lay the groundwork for our later unit on **multiplexing**.

2 Media, Signals, Data

2.1 Media

Network signals might be propagated through **guided** media such as metal wires (twisted pairs, coaxial cable, etc.) or optical fibers (flashing light, off for 0, on for 1), or in free space, such as with radio waves or microwaves.

2.2 Signal Types

A signal may be in **digital** form, e.g. 0-1 bits with a low voltage meaning 0 and a high voltage meaning 1. The graph of a digital signal over time has a "square wave" shape.

On the other hand, **analog** forms look "wavy." A modem, for instance, sends a low-pitched sound for a 0 and a high pitch for a 1; the sounds themselves look like sine waves when graphed against time.

2.3 Data Types

Just as the signals of interest can be either digital or analog, the data encoded by those signals could be either digital, such as numbers or characters, or analog, such as voices.

Either type of data can be encoded within either type of signal. We noted above, for instance, that a modem encodes digital data onto analog signals. Voice data, on the other hand, though inherently analog, can be **digitized**: A person's voice graphed against time is a continuous curve, but we can sample it at regular intervals, typically 8,000 times per second. At each sample point, we record the numerical height of the curve (i.e. the numerical value of the loudness). This gives us numbers, which comprise digital data. These can be stored in a disk file as with a game program, sent through the Internet as with Real Audio transmission of radio programs, and so on.

3 Spectral Analysis

3.1 Fourier Series

Recall from calculus that we can write a function x(t) as a Taylor series, which is an "infinite polynomial":

$$x(t) = \sum_{n=0}^{\infty} c_n t^n \tag{1}$$

For instance, for e^t,

$$e^t = \sum_{n=0}^{\infty} \frac{1}{n!} t^n \tag{2}$$

Recall also that we say a function x(t) (where t is time) is **periodic** with period T if if x(u+T) = x(u) for all u. The **fundamental frequency** of x is then defined to be the number of periods per unit time,

$$f_0 = \frac{1}{T} \tag{3}$$

Suppose x(t) is a digital or analog periodic signal transmitted in some medium. We can write x as an “infinite trig polynomial,” i.e. a **Fourier series**:

$$x(t) = \sum_{n=0}^{\infty} a_n \cos(2\pi n f_0 t) + \sum_{n=1}^{\infty} b_n \sin(2\pi n f_0 t) \tag{4}$$

(The second summation starts at n = 0 too, but the sine term in that case is zero.) Here, instead of having a series of terms

$$1,\ t,\ t^2,\ t^3,\ ... \tag{5}$$

as in a Taylor series, we have a series of terms

$$1,\ \cos(2\pi f_0 t),\ \cos(4\pi f_0 t),\ \cos(6\pi f_0 t),\ ... \tag{6}$$

and similar sine terms. Note that the frequences in those cosines are integer multiples, called **harmonics**, of the fundamental frequency of x, f_0.

The set of coefficients $\{a_n\}$, $\{b_n\}$ is called the **frequency spectrum** of x. The coefficients are calculated as follows:

$$a_0 = \frac{1}{T} \int_0^T x(t) dt \tag{7}$$

$$a_n = \frac{2}{T} \int_0^T x(t) cos(2\pi n f_0 t) dt \tag{8}$$

$$b_n = \frac{2}{T} \int_0^T x(t) sin(2\pi n f_0 t) dt \tag{9}$$

Because of the periodic nature of the functions involved, we can shift the range of integration by equal amounts on the lower and upper bounds, and it is often convenient to do so if we are calculating the integrals by hand. Any lower and upper bounds which differ by the amount T will give the same answer.

We say that x(t) is the **energy level** of the signal. For example, if x(t) is a graph of your voice over time, x(t) is the loudness of your voice at time t. The a_n and b_n then show how the energy of the signal break down into different frequencies; in fact, the average squared energy of the signal is the sum of the squares of these coefficients:

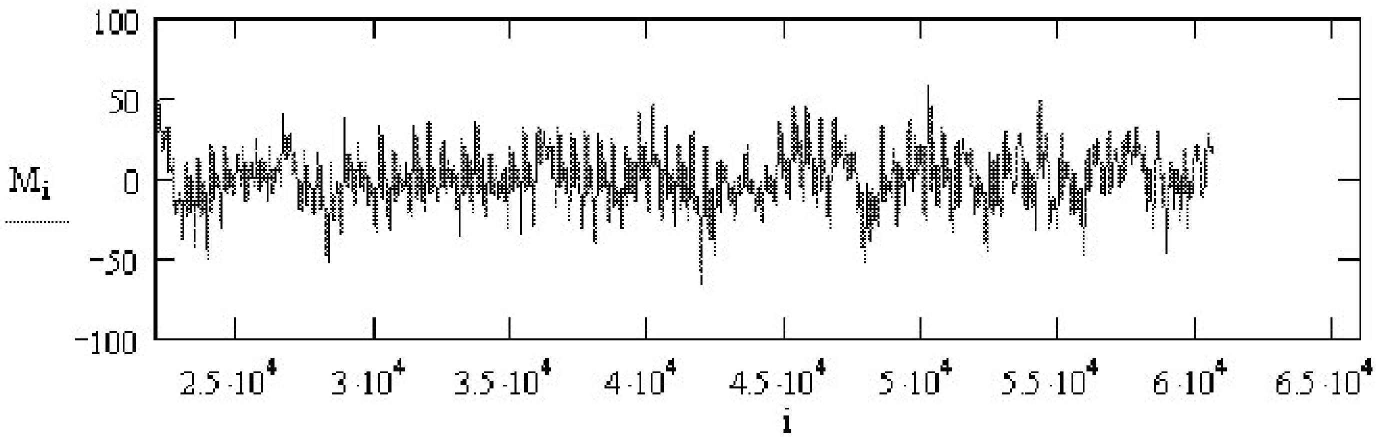
$$\frac{1}{T}\int_0^T x^2(t)dt = a_0^2 + \frac{1}{2}\sum_{n=1}^{\infty}(a_n^2 + b_n^2)$$

We say that x(t) is the **time domain** version of the signal, and a_n and b_n comprise the **frequency domain**.

We can also write x as an integral of trig functions, rather than a sum of such functions. Then the spectrum is a continuous range of numbers, rather than the discrete points a_n and b_n. This is called the **Fourier Transform** of the original periodic function.

3.2 Example: Time- and Frequency-Domain Graphs for a Vibrating Reed

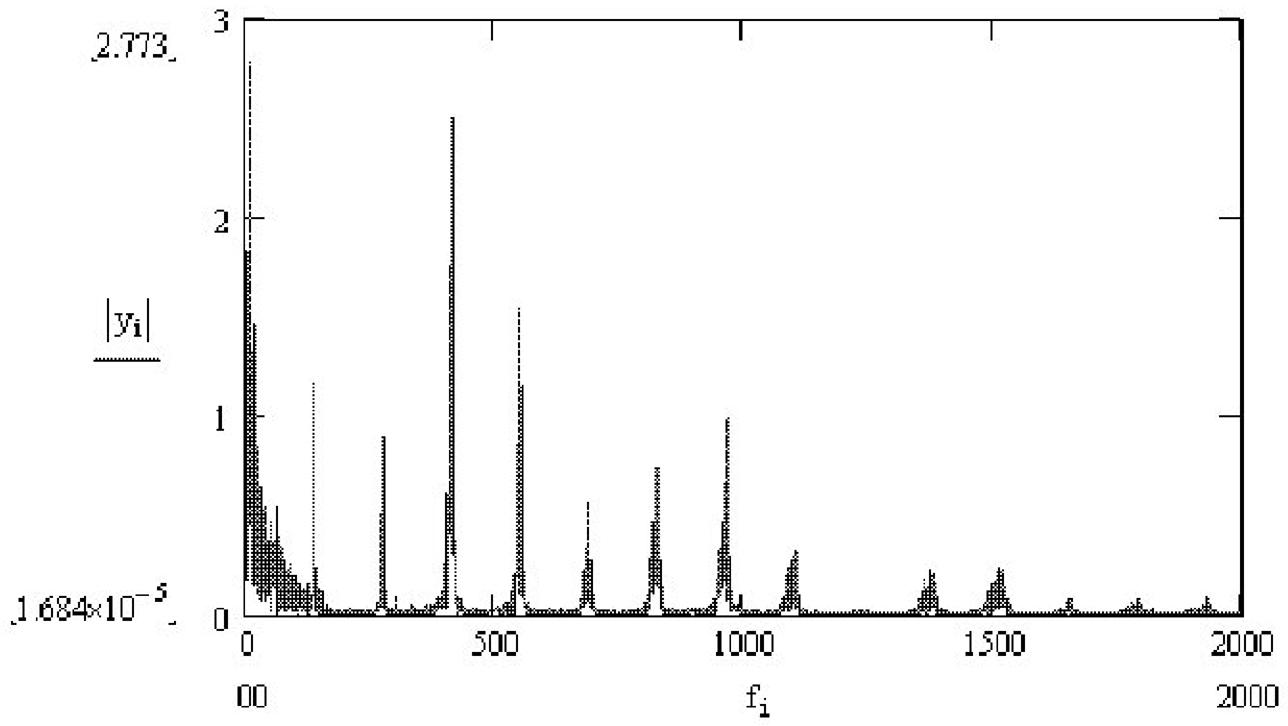
Here is a time-domain graph of the sound made by a vibrating reed:[1]



Now here is the frequency-domain version:[2]

[1]Reproduced here by permission of Prof. Peter Hamburger, Indiana-Purdue University, Fort Wayne. See http://www.ipfw.edu/math/Workshop/PBC.html

[2]The vertical axis represents the Fourier-transform analog of $(a_n^2 + b_n^2)^{0.5}$.



Note that this graph is very "spiky." In other words, most of the power of the signal is at a few frequencies which arise from the physical properties of a reed. By contrast, you would not see this spiky behavior in a graph for the human voice, which is much more complex.

3.3 Application to Transmitted Signals

Suppose for example x(t) is the loudness of your voice at time t, and we are transmitting it (without change) through some medium. Your voice is a sum of components at various frequencies, and a_n and b_n in Equation (4) represent the loudness of the n-th frequency.

The same is true if we are transmitting data, say the bit sequence 10110101..., with x(t) being of square-wave form. We can still represent x(t) as a Fourier series.

Of course, all this assumes a periodic signal. But strictly speaking, the signals we send are not periodic. Consider, for example, a text file transfer, transmitted as a digital signal. The characters in the file do not follow a periodic pattern, and thus the bits we send do not follow a periodic pattern, and the graph of the signal against time will not be periodic. However, for the purpose of analyzing the speed capacity of a given physical medium to transmit the data, we can imagine the signal to be periodic. If for instance the text file includes the sentence "The quick brown fox jumped," we can ask how fast the medium could transmit the corresponding signals if the sentence were to be transmitted again and again (resulting in periodic signals). And for voice data transmitted in analog form, for instance, the signals are periodic during short time intervals.

3.4 Independence of Bit Rate

Note that if we are transmitting data, the a_n and b_n are independent of our bit rate, i.e. the number of bits per second we transmit. Here is how to see this:

Suppose for example that we are sending the periodic data sequence 110101101011010... (i.e. we repeatedly send the bit pattern 11010) at a rate of c bits per second. Then each bit takes 1/c seconds to send. The waveform x(t) would have the value 1 between 0 and 1/c, 1 again between 1/c and 2/c, then have the value 0 between 2/c and 3/c, etc. , and T = 5/c.

Now suppose we were to send the same signal but at half the bit rate, say because we changed modems. Call the new signal y(t) and the new period T'. Then y(t) = x(0.5t), and T' = 2T. The period in terms of numbers of bits, 5, has not changed, but the time to send those bits has doubled.

Then

$$a_n = \frac{2}{T'} \int_0^{T'} y(t) cos(2\pi n \frac{1}{T'} t) dt$$

If one uses the facts that y(t) = x(0.5t), and T' = 2T, and then one does the change-of-variable u = 0.5t, we find we get the same integral for a_n for y(t) as for x(t), and a similar statement would hold for b_n.

The intuition here is that although the graph of the curve y(t) would be a "stretched out" version of x(t), it would still have the same shape, and the same relative weighting would occur for all the sine and cosine terms (since they are "stretched out" too).

3.5 Bandwidth: How to Read the *San Francisco Chronicle* Business Section

The popular press, especially business or technical sections, often uses the term **bandwidth**. What does this mean?

Any transmission medium has a natural range [f_{min},f_{max}] of frequencies that it can handle well. For example, an ordinary voice-grade telephone line can do a good job of transmitting signals of frequencies in the range 0 Hz to 4000 Hz, where "Hz" means cycles per second. Signals of frequencies outside this range suffer fade in strength, i.e are **attenuated**, as they pass through the phone line. We call the frequency interval [0,4000] the **effective bandwidth** (or just the **bandwidth**) of the phone line.

In addition to the bandwidth of a **medium**, we also speak of the bandwidth of a **signal**. For instance, although your voice is a mixture of many different frequencies, represented in the Fourier series for your voice's waveform, the really low and really high frequency components, outside the range [340,3400], have very low power, i.e. their a_n and b_n coefficients are small. Most of the power of your voice signal is in that range of frequencies, which we would call the effective bandwidth of your voice waveform.[3]

Obviously, in order for your voice to be heard well on the other end of your phone connection, the bandwidth of the phone line must be at least as broad as that of your voice signal, and that is the case. However, the phone line's bandwidth is not much broader than that of your voice signal. So, some of the frequencies in your voice will fade out before they reach the other person, and thus some degree of distortion will occur. It is common, for example, for the letter 'f' spoken on one end to be mis-heard as 's'on the other end. This

[3]This is also the reason why digitized speech is sampled at the rate of 8,000 samples per second. A famous theorem, due to Nyquist, shows that the sampling rate should be double the maximum frequency. Here the number 3,400 is "rounded up" to 4,000, and after doubling we get 8,000.

also explains why your voice sounds a little different on the phone than in person. Still, most frequencies are reproduced well and phone conversations work well.

We often use the term "bandwidth" to literally refer to width, i.e. the width of the interval [f_{min},f_{max}], $f_{max} - f_{min}$.

There is huge variation in bandwidth among transmission media. As we have seen, phone lines have bandwidth intervals covering values on the order of 10^3. For optical fibers, these numbers are more on the order of 10^{15}. Suppose that for a given medium $f_{min} = 0$ and f_{max} = N f_0. Then the signal

$$x(t) = \sum_{n=0}^{\infty} a_n \cos(2\pi n f_0 t) + \sum_{n=1}^{\infty} b_n \sin(2\pi n f_0 t) \tag{10}$$

is in effect truncated by the transmission medium to

$$x_N(t) = \sum_{n=0}^{N} a_n \cos(2\pi n f_0 t) + \sum_{n=1}^{N} b_n \sin(2\pi n f_0 t) \tag{11}$$

i.e. an error is made and x(t) is distorted. The larger N is, the less distortion. For any given signal, the higher f_{max} of the transmission medium, the less distortion.[4]

Now from calculus, for any given signal x(t), $a_n, b_n \rightarrow 0$ as $n \rightarrow \infty$. Thus, the a_n and b_n get small after a while, say, after a_M and b_M. So,

$$x(t) \approx \sum_{n=0}^{M} a_n \cos(2\pi n f_0 t) + \sum_{n=1}^{M} b_n \sin(2\pi n f_0 t) \tag{12}$$

In other words, the effective bandwidth of the signal is [0,Mf_0]. As long as we send x(t) along a transmission medium for which $N \geq M$, x(t) will be received with reasonable fidelity.

3.6 Effect of Bandwidth on Transmission Rates

In order to get an idea of the relation of bandwidth to transmission rate, suppose we are sending the bit pattern 101010101010..., and that in our transmission medium a level (say volts, tenths of volts, or whatever) of +1 represents a 1 bit and a -1 represents a 0. Note that we are just using 101010101010... as an example; our transmission medium must be able to send all bit patterns. But we will use this one to illustrate the issues involved, so we can see the how the bandwidth of our transmission medium limits how fast we can send bits.

Suppose we wish to send at the rate of 2×10^6 bits per second. What will be the period for the signal 101010101010 above? In terms of bit times, the period is two bit times. How much is that in seconds? At our 2 million bits/sec rate, each bit takes 0.5 millionths of a second, i.e. 0.5 microsecond. Thus our period is T = 1 microsecond, and the fundamental frequency f_0 is 1 megaherz (MHz), i.e. 1 million cycles per second. (Recall that $f_0 = 1/T$.)

[4]level of quality we consider acceptable for a received signal, the larger N is, the more signals we can **multiplex** onto the given medium. This will be explored later.

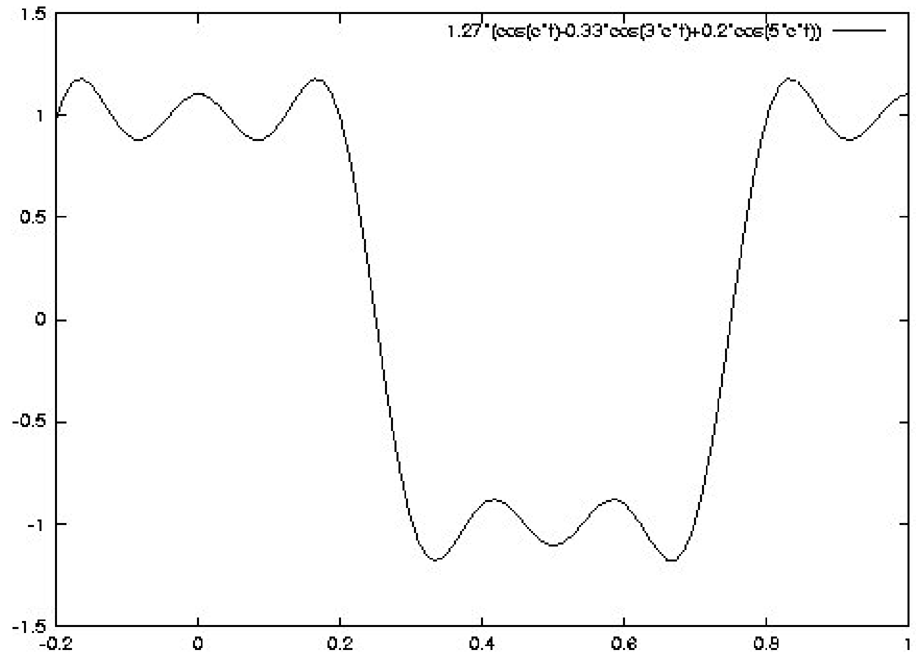
The period consists of two bit times, specifically one 1 bit and one 0 bit, each taking T/2 time to transmit. We could describe x(t) as having the value 1 from 0 to $\frac{T}{2}$, the value -1 from $\frac{T}{2}$ to T, and so on, repeating in this manner from $-\infty$ to ∞.

Using the formulas, we find that: all b_n are 0; all even-numbered a_n are 0; and a_1, a_3, a_5 and so on are $\frac{4}{\pi}, -\frac{4}{3\pi}, \frac{4}{5\pi}$, et cetera. Note that since f_0 is 1 MHz, these nonzero coefficients will be for cosine terms with frequencies 1 MHz, 3 MHz, 5 MHz, et cetera.

Now suppose our transmission medium only transmits frequencies up to 5 MHz. Then even though we place onto our transmission medium x(t)—a squarewave with values 1, -1, 1, -1, etc.—the medium will distort x(t) to the function

$$\frac{4}{\pi}[cos(2\pi \cdot f_0 t) - \frac{1}{3}cos(2\pi \cdot 3f_0 t) + \frac{1}{5}cos(2\pi \cdot 5f_0 t)] \tag{13}$$

Using microseconds for our horizontal axis (since T = 1 microsecond), a plot of this series would look like this:



So, even though x(t) is distorted, the result is not a bad approximation to the original waveform! We could design circuitry that would be able to distinguish well between a 1 bit and a 0 bit in this case.[5] For example, we could design the receiver to treat any signal which is greater than 0.8 during 80% of a bit time as a 1, and any signal which is less than -0.8 during 80% of a bit time as a -1.

In other words, a medium which transmitted all frequencies up to 5 MHz would be able to successfully send at the proposed rate of 2,000,000 bits per second, at least for the bit pattern we have looked at here, 101010...

The point of all this is that the bandwidth of the transmission medium determines the maximum bit rate at which one can send. Suppose for instance that the transmission medium only passes frequencies up to 4 MHz. Then if we were to send 101010... at the proposed rate of 2,000,000 bits per second, we would only get two cosine terms, and thus a much poorer approximation to x(t), probably so poor that it would

[5]Of course, this is not accounting for other factors, such as further distortion of the waveform, such as by line noise.

produce bit-recognition errors. In order to get three cosine terms (which we found was sufficient to keep x(t) from getting distorted too much), we would be forced to transmit at a lower rate.

How much lower? Well, we would need to have $5f_0 = 4000000$, so we would need T = 1.25 microseconds. Since each period consists of two bit times, that would be 0.625 microseconds per bit, thus a bit rate of 1,600,000 bits per second, considerably lower than our hoped-for 2,000,000 bits per second.

Note even though the a_n and b_n are independent of the bit rate, f_0 does implicitly incorporate the bit rate. Say that a period consists of b bits. For example, b = 2 in our 101010... example above, since the 2-bit pattern 10 repeats, while 101110111011 would have b = 4, since the 4-bit pattern 1011 repeats. Say we send bits at the rate of s bits per second, so each bit takes time 1/s. Then T = b x (1/s), so

$$f_0 = s/b \tag{14}$$

4 Bit Encodings

Where does one bit start and the next begin? For instance, suppose our line has a high voltage for a 1 and a low voltage for a 0, and suppose that our sending device were to send 10 1s in a row. Then the line would be high for 10 bit times, but how would the receiver know that this is not, for example, only nine 1s? There is no “boundary” between bits.

Theoretically speaking, this problem is solved by having a clock on the receiver end, carefully marking off bit times. If we are sending at, say, 10,000 bits per second, that would mean that each bit takes 100 millionths of a second, i.e. 100 microseconds. So if the receiver sees the line is high for 1000 microseconds, then the receiver knows that this must be 1000/100 = 10 1 bits.

The problem with this is that there will be discrepancies between the sender and receiver clocks. Over time, this discrepancy will accumulate and eventually the receiver will be off a full bit time from the sender—and thus the receiver will look at the wrong bit.

In order to deal with this, we design the receiver to frequently resynchronize itself with the sender. The way this is typically done is to have the receiver look for **line transitions**, i.e. changes from 1 to 0 or vice versa. Since a transition occurs between two consecutive bits, the receiver can use this to resynchronize its clock with that of the sender.

But if we happen to send a long string of 1s, the receiving clock won’t have this chance to resynchronize. One method of dealing with this is **non-return-to-zero-inverted** (NRZI) coding, which works as follows. If we wish to send a 1 bit, we place a voltage opposite to what we sent in the previous bit. In other words, if the previous line level had been low, we now put a high value on the line, and vice versa. If we wish to send a 0, we send the same voltage level as in the previous bit.

The idea here is to force frequent line transitions. This scheme does this, since every 1 bit will be coded as a line transition. Thus the receiver has frequent opportunities to resynchronize its clock.

That solves the problem of having a long string of 1s in our message, but it doesn’t cover cases in which we happen to send long strings of 0s. Both problems are addressed by **Manchester** coding, which forces a bit transition in *each* bit time. Under this system, if we wish to send a 0 bit, we hold the line at low voltage for one transmission bit time (which will be half a data bit time) and then bring it up to high voltage for one transmission bit time. If we wish to send a 1 bit, we have the line high for a transmission bit time and then low for a transmission bit time. This guarantees a line transition during each data bit, so the receiver has a chance to resynchronize its clock during every data bit.

But Manchester is wasteful—each data bit now needs two transmission bit times to send. Thus for any given transmission line, and thus for any given maximum line bandwidth, and thus for any given maximum transmission bit rate, it cuts down the allowable data bit transmission rate by half.

Note that Manchester coding also changes the pattern in which we uses our transition line's frequency bandwidth. For instance, recall our earlier example, in which we were transmitting at 2 MHz, and our message consisted of 101010... Using Manchester coding, the values on our line would be 100110011001..., which has a quite different Fourier series than would the 101010... pattern we would use without Manchester coding.

So, compromise schemes have been developed, such as the famous 4B/5B system. Under this scheme, each group of four message bits is coded in five bit times. Here is the translation table:

data	coding
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

The five-bit code is then sent using NRZI. For example, suppose we wish to send 1011, and that the last state of the line had been high. Then we would send 1011 as low-low-high-low-high. (Make sure you understand this point.)

Note that in all the five-bit codes above, there is never more than one leading 0 and never more than two trailing 0s. So, if two data items are sent back-to-back, we cannot have more than three consecutive 0s. This solves the original NRZI problem concerning long strings of 0s discussed earlier, and since NRZI itself handles the problem of long strings of 1s, you can see that the 4B/5B scheme works pretty well.