

CDMA with Direct Sequence Spread Spectrum

Jie Gao*

February 13, 2007

1 DSSS Overview

The idea of CDMA is to use different codes to distinguish different users. In DSSS, each user has a chipping sequence (or called code, key, etc) that determines how data is encoded and decoded. Here is the protocol on both the sender and receiver side.

Sender. Take each bit of the user data, XOR¹ with the chipping sequence (e.g., 10110111000). Essentially corresponding to a data bit of 1, we have a sequence of 11 bits 10110111000. Corresponding to 0 in the user data, we have the chipping sequence with all bits flipped, 01001000111.

Receiver. Take the received message, XOR with the chipping sequence. Then sum up every 11 bits. If the sum is smaller or equal to 5, then take that bit as a 0; else take the bit as 1.

2 CDMA

In a system with multiple users transmitting at the same time. The signals are overlaid. In other words, if one does not know the code, then it hears some signal like random noise. However, for the receiver with the correct code, it will be able to extract from the noise-like data its expected information.

The issue of the system design is to design the codes for different users such that they can successfully extract their respective data. To explain the criterion for code design in CDMA, we first use the language of vectors and re-iterate the communication protocol.

Each chipping sequence, or code, is a sequence of 0's and 1's. We will represent that by a vector. Essentially replace 1 by a corresponding 1 and replace 0 by a corresponding -1. For example, the Baker code 10110111000 is represented by a vector of dimension 11, represented by

$$(1, -1, 1, 1, -1, 1, 1, 1, -1, -1, -1).$$

Similarly, we also represent a signal bit of 1 by 1 and a data bit 0 by -1.

Now the XOR of two bits is exactly multiplication in this new system. The XOR of the chipping sequence of the input signal is exactly the bit-wise product in the vector space. The extraction of the data is the *dot product* of two vectors: take the bit-wise product and sum up all the values.

Here is the same communication protocol as the previous section, but explained in the language of vector operations.

*Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA, jgao@cs.sunysb.edu.

¹Recall that XOR outputs 1 if the two inputs are different and outputs 0 if the two inputs are the same.

The sender A has data $A_d = 1$ (stands for bit 1). It will take the product of A_d with the chipping sequence $A_k = 010011 = (-1, 1, -1, -1, 1, 1)$:

$$A_d \cdot A_k = A_d \times (-1, 1, -1, -1, 1, 1).$$

The sender B has data $B_d = -1$ (stands for bit 0), and takes takes the product of B_d with the chipping sequence $B_k = 110101 = (1, 1, -1, 1, -1, 1)$, which is:

$$B_d \cdot B_k = B_d \times (1, 1, -1, 1, -1, 1).$$

Since both transmission happen at the same time, the two signals will superimpose on each other. Thus the receiver of A and B will hear the following signal:

$$S = A_d \cdot A_k + B_d \cdot B_k = (0, 2, -2, 0, 0, 2).$$

Now here is the key point. With the code A_k , a user can decipher from the superimposed signal the data sent by A . Basically it XOR its code with the signal, or in other words, performs a dot product of the code A_k with the signal S :

$$A_k \cdot S = (-1, 1, -1, -1, 1, 1) \cdot (0, 2, -2, 0, 0, 2) = 6.$$

With the code of B_k , a user can decipher the data sent by B , again, by doing a dot product of B_k with S :

$$B_k \cdot S = (1, 1, -1, 1, -1, 1) \cdot (0, 2, -2, 0, 0, 2) = -6.$$

A value larger than 0 translates to 1 and a value smaller than 0 translates to 0. Now we can see the decoding is successful.

3 Criterion for Code Design

Now we should design the codes for different users such that decoding in the previous section is successful. The criteria are the following:

- A code should have a good autocorrelation, $A_k \cdot A_k$ is large;
- A code should be orthogonal to other codes, $A_k \cdot B_k$ is zero.

To show why a code with this property is a good code, let's go back to the decoding phase. We can verify that the two codes we use are good. $A_k \cdot A_k = B_k \cdot B_k = 6$ and $A_k \cdot B_k = 0$. In order to decode A_d , we do the following.

$$\begin{aligned} A_k \cdot S &= A_k \cdot (A_d \cdot A_k + B_d \cdot B_k) \\ &= A_d \cdot (A_k \cdot A_k) + A_d \cdot (A_k \cdot B_k) \\ &= 6A_d \end{aligned}$$

Similarly,

$$\begin{aligned} B_k \cdot S &= B_k \cdot (A_d \cdot A_k + B_d \cdot B_k) \\ &= A_d \cdot (A_k \cdot B_k) + B_d \cdot (B_k \cdot B_k) \\ &= 6B_d \end{aligned}$$

For the above example we assume that the two senders A, B have the same signal strength. In reality it may just be possible that one has a higher signal strength. Say B is 5 times louder. Now we have

$$S = A_d \cdot A_k + 5B_d \cdot B_k.$$

The observation is that this does not affect the decoding process at all. We still have

$$\begin{aligned}A_k \cdot S &= A_k \cdot (A_d \cdot A_k + 5B_d \cdot B_k) \\ &= A_d \cdot (A_k \cdot A_k) + 5A_d \cdot (A_k \cdot B_k) \\ &= 6A_d\end{aligned}$$

and

$$\begin{aligned}B_k \cdot S &= B_k \cdot (A_d \cdot A_k + 5B_d \cdot B_k) \\ &= A_d \cdot (A_k \cdot B_k) + 5B_d \cdot (B_k \cdot B_k) \\ &= 30B_d\end{aligned}$$

The signal recovered from sender B is also louder.

Autocorrelation of a code helps with synchronization. At the receiver side it needs to know the starting point of the code. One can simply take the code A_k and perform XOR with every string of the same length. The larger the value $A_k \cdot A_k$, the better to distinguish the data — at some point receiver can detect a sufficiently high signal strength, which indicates the starting point of the code is found.